SIEMENS

SIMATIC HMI

WinCC (TIA Portal)

System Manual

11/2022

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

Ⓟ 11/2022 Subject to change

Table of contents

Installation 1

1.1 Notes on the installation

Content

Information that could not be included in the online help and important information about product characteristics.

Installation in a virtual environment (private cloud)

You can find instructions on how to install TIA Portal in a virtual environment (private cloud) on the installation data medium in the directory "Documents\Readme\<language directory>". You can open the PDF document "TIAPortalCloudConnectorHowTo<language ID>.pdf" here.

Automatic installation

A description of automated installation is available on the product DVD in the directory "Documents\Readme\<Language directory>".

Use of the same versions of TIA Portal products during installation

When installing different TIA Portal products, make sure that you use the same versions of service packs and updates for the installation. For example, if you have installed SP1 for STEP 7 V13, you must also install SP1 for WinCC V13. The service packs and updates must be installed for all products at the same time. Do not start TIA Portal until all products have been upgraded.

You can download the service packs from the Internet under Siemens Industry Online Support (https://support.industry.siemens.com/cs/de/en/).

Target directory of the installation

Do not use any UNICODE characters (for example, Chinese characters) in the installation path.

The installation path cannot be changed after installing one of the TIA Portal products.

Security settings for the installation of WinCC

When you install WinCC, security settings are changed in your operating system. These security settings are listed during the installation.

You must confirm the changes to the security settings.

If you make changes to your operating system after installing it, the security settings can be changed by installing TIA Portal.

You can restore the security settings by installing TIA Portal as follows:
"Start > All Programs > Siemens Automation > Security Controller > Restore settings".

Use of antivirus programs

During the installation, read and write access to already installed files is necessary. Some antivirus programs block this access. Therefore, configure your antivirus program for the installation of TIA Portal so that access to these files is possible.

Installation of PCT V3.5.2

PCT V3.5.1 is automatically uninstalled with the installation of TIA Portal V18.

If necessary, install PCT V3.5.2 manually from DVD 2.

FAQs on TIA Portal

FAQs on TIA Portal are available under FAQs.

1.2 Licensing

1.2.1 Notes on licenses

Availability of licenses

The licenses for the products of the TIA Portal are shipped on an installation data medium or via online software delivery (OSD).

Before you uninstall the TIA Portal, you must transfer and back up the licenses still required. Use the Automation License Manager for this purpose.

Provision of the Automation License Manager

The Automation License Manager is supplied on the installation data medium and is transferred automatically during the installation process.

If you uninstall the TIA Portal, the Automation License Manager remains installed on your system.

Working with the Automation License Manager

The Automation License Manager is a product of Siemens AG, which is used for handling license keys (technical representatives of licenses).

Software products that require license keys for operation, such as the TIA Portal, register the need for license keys automatically with the Automation License Manager . If the Automation License Manager finds a valid license key for this software, the software can be used according to the license usage terms associated with this license key.

Note

For additional information on how to manage your licenses with the Automation License Manager , refer to the documentation supplied with the Automation License Manager .

1.2.2 Licensing STEP 7 and WinCC

Introduction

You require a License Key to license the following STEP 7 editions:

- STEP 7 Basic
- STEP 7 Professional

You can install the corresponding License Key for STEP 7 using the Automation License Manager after the installation has been completed.

There are other licenses available for the optional packages of STEP 7. You can get detailed information in the documentation of the corresponding option package.

You can transfer the corresponding license keys for STEP 7 and STEP 7 options after the installation with the Automation License Manager.

For licensing the following editions in WinCC, you need a License Key:

- WinCC Engineering System
- WinCC Runtime
- Options for WinCC Engineering System
- Options for WinCC Runtime system

You transfer licenses for WinCC and the WinCC add-ons after installation with the Automation License Manager.

Note

The licensee also recognizes that the software (SW) from the Microsoft Corporation or subsidiaries contains licensed software. The licensee hereby agrees to be bound by and comply with the terms of the attached license agreement between Microsoft SQL Server and End User.

Licenses

The following licenses with the corresponding License Keys are available:

- STEP 7 Basic
- STEP 7 Professional
- STEP 7 Professional Combo

- WinCC Basic
- WinCC Comfort
- WinCC Comfort Combo
- WinCC Advanced
- WinCC Advanced Combo
- WinCC Professional
- WinCC Professional Combo
- WinCC Unified Basic Engineering
- WinCC Unified Comfort Engineering
- WinCC Unified PC Engineering
- WinCC RT Advanced
- WinCC RT Professional
- WinCC Unified PC Runtime

Usage possibilities of licenses and devices

The following table shows which devices you can use with which license:

Existing license	**Devices that can be used**						
	Unified Devices			Classic Devices			
	WinCC Unified PC	Comfort Panels	Basic Panels	RT Professional	RT Advanced	Comfort Panels	Basic Panels
WinCC Unified Basic ES**	No	No	Yes	No	No	No	Yes
WinCC Unified Comfort ES	No	Yes	Yes	No	No	Yes	Yes
WinCC Unified PC 10k ES	Yes (10k*)	Yes	Yes	No	Yes	Yes	Yes
WinCC Unified PC 100k ES	Yes (100k*)	Yes	Yes	Yes	Yes	Yes	Yes
WinCC Unified PC max ES	Yes	Yes	Yes	Yes	Yes	Yes	Yes
WinCC Basic	No	No	Yes	No	No	No	Yes
WinCC Comfort	No	Yes	Yes	No	No	Yes	Yes
WinCC Advanced	Yes (10k*)	Yes	Yes	No	Yes	Yes	Yes
WinCC Professional 512	Yes (10k*)	Yes	Yes	Yes (512*)	Yes	Yes	Yes
WinCC Professional 4096	Yes (10k*)	Yes	Yes	Yes (4096*)	Yes	Yes	Yes
WinCC Professional max	Yes (100k*)	Yes	Yes	Yes	Yes	Yes	Yes

* Maximum possible number of PowerTags

** Released with HSP.

Validity of license keys for older versions of STEP 7

With a valid License Key for V18.x of STEP 7 Professional and STEP 7 Professional Combo, you can also operate older versions of STEP 7 without restrictions. The following table provides more detailed information about this:

Edition	License	Valid for
STEP 7 Basic V18.x	STEP 7 Basic	• STEP 7 Basic V18.x • STEP 7 Basic V17.x • STEP 7 Basic V16.x • STEP 7 Basic V15.x • STEP 7 Basic V14.x • STEP 7 Basic V13.x • STEP 7 Basic V12.x • STEP 7 Basic V11.x
STEP 7 Professional V18.x	STEP 7 Professional	• STEP 7 Professional V18.x • STEP 7 Professional V17.x • STEP 7 Professional V16.x • STEP 7 Professional V15.x • STEP 7 Professional V14.x • STEP 7 Professional V13.x • STEP 7 Professional V12.x • STEP 7 Professional V11.x • STEP 7 Basic V18.x • STEP 7 Basic V17.x • STEP 7 Basic V16.x • STEP 7 Basic V15.x • STEP 7 Basic V14.x • STEP 7 Basic V13.x • STEP 7 Basic V12.x • STEP 7 Basic V11.x

Edition	License	Valid for
STEP 7 Professional V18.x	STEP 7 Professional Combo	• STEP 7 Professional V18.x • STEP 7 Professional V17.x • STEP 7 Professional V16.x • STEP 7 Professional V15.x • STEP 7 Professional V14.x • STEP 7 Professional V13.x • STEP 7 Professional V12.x • STEP 7 Professional V11.x • STEP 7 Basic V18.x • STEP 7 Basic V17.x • STEP 7 Basic V16.x • STEP 7 Basic V15.x • STEP 7 Basic V14.x • STEP 7 Basic V13.x • STEP 7 Basic V12.x • STEP 7 Basic V11.x • STEP 7 V5.6 • STEP 7 V5.5 • STEP 7 V5.4 • STEP 7 Professional 2017 • STEP 7 Professional 2010 • STEP 7 Professional 2006

Validity of license keys for older versions of WinCC

With a valid License Key for WinCC V18.x, you can also operate older versions of WinCC without restrictions.

You can find more detailed information in the following table:

WinCC Engineering System

Edition	License	Valid for
WinCC Basic V18.x	WinCC Basic	• WinCC Basic V18.x • WinCC Basic V17.x • WinCC Basic V16.x • WinCC Basic V15.x • WinCC Basic V14.x • WinCC Basic V13.x • WinCC Basic V12.x • WinCC Basic V11.x • WinCC Unified Basic V18.x*

Edition	License	Valid for
WinCC Comfort/Advanced V18.x	WinCC Comfort	• WinCC Basic, Comfort V18.x • WinCC Basic, Comfort V17.x • WinCC Basic, Comfort V16.x • WinCC Basic, Comfort V15.x • WinCC Basic, Comfort V14.x • WinCC Basic, Comfort V13.x • WinCC Basic, Comfort V12.x • WinCC Basic, Comfort V11.x • WinCC Unified Basic V18.x*
	WinCC Comfort Combo	• WinCC flexible Compact, Standard >= 2005 • WinCC Basic, Comfort V18.x • WinCC Basic, Comfort V17.x • WinCC Basic, Comfort V16.x • WinCC Basic, Comfort V15.x • WinCC Basic, Comfort V14.x • WinCC Basic, Comfort V13.x • WinCC Basic, Comfort V12.x • WinCC Basic, Comfort V11.x • WinCC Unified Basic V18.x*
	WinCC Advanced	• WinCC Basic, Comfort, Advanced V18.x • WinCC Basic, Comfort, Advanced V17.x • WinCC Basic, Comfort, Advanced V16.x • WinCC Basic, Comfort, Advanced V15.x • WinCC Basic, Comfort, Advanced V14.x • WinCC Basic, Comfort, Advanced V13.x • WinCC Basic, Comfort, Advanced V12.x • WinCC Basic, Comfort, Advanced V11.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x • WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x
	WinCC Advanced Combo	• WinCC flexible Compact, Standard, Advanced >= 2005 • WinCC Basic, Comfort, Advanced V18.x • WinCC Basic, Comfort, Advanced V17.x • WinCC Basic, Comfort, Advanced V16.x • WinCC Basic, Comfort, Advanced V15.x • WinCC Basic, Comfort, Advanced V14.x • WinCC Basic, Comfort, Advanced V13.x

Edition	License	Valid for
		• WinCC Basic, Comfort, Advanced V12.x • WinCC Basic, Comfort, Advanced V11.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x • WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x
WinCC Professional V18.x	WinCC Professional (512) WinCC Professional (4096) WinCC Professional (max.)	• WinCC Basic, Comfort, Advanced, Professional V18.x • WinCC Basic, Comfort, Advanced, Professional V17.x • WinCC Basic, Comfort, Advanced, Professional V16.x • WinCC Basic, Comfort, Advanced, Professional V15.x • WinCC Basic, Comfort, Advanced, Professional V14.x • WinCC Basic, Comfort, Advanced, Professional V13.x • WinCC Basic, Comfort, Advanced, Professional V12.x • WinCC Basic, Comfort, Advanced, Professional V11.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x • WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x
	WinCC Professional (512) Combo WinCC Professional (4096) Combo WinCC Professional (max.) Combo	• WinCC flexible Compact, Standard, Advanced >= 2008 SP3 • WinCC Basic, Comfort, Advanced, Professional V18.x • WinCC Basic, Comfort, Advanced, Professional V17.x • WinCC Basic, Comfort, Advanced, Professional V16.x • WinCC Basic, Comfort, Advanced, Professional V15.x • WinCC Basic, Comfort, Advanced, Professional V14.x • WinCC Basic, Comfort, Advanced, Professional V13.x • WinCC Basic, Comfort, Advanced, Professional V12.x • WinCC Basic, Comfort, Advanced, Professional V11.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x • WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x

Edition	License	Valid for
WinCC Unified Basic V18.x*	WinCC Unified Basic ES	• WinCC Basic V18.x • WinCC Unified Basic V18.x*
WinCC Unified Comfort V18.x	WinCC Unified Comfort ES	• WinCC Basic, Comfort V18.x • WinCC Basic, Comfort V17.x • WinCC Basic, Comfort V16.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x
WinCC Unified PC V18.x	WinCC Unified PC (10k) ES	• WinCC Basic, Comfort, Advanced V18.x • WinCC Basic, Comfort, Advanced V17.x • WinCC Basic, Comfort, Advanced V16.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x • WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x
	WinCC Unified PC (100k) ES WinCC Unified PC (max.) ES	• WinCC Basic, Comfort, Advanced, Professional V18.x • WinCC Basic, Comfort, Advanced, Professional V17.x • WinCC Basic, Comfort, Advanced, Professional V16.x • WinCC Unified Basic V18.x* • WinCC Unified Comfort V18.x • WinCC Unified Comfort V17.x • WinCC Unified Comfort V16.x • WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x

* Released with HSP.

WinCC Runtime

Edition	License	Valid for
WinCC RT Advanced V18.x	WinCC RT Advanced (16384) WinCC RT Advanced (8192) WinCC RT Advanced (4096) WinCC RT Advanced (2048) WinCC RT Advanced (512) WinCC RT Advanced (128)	• WinCC RT Advanced V17.x • WinCC RT Advanced V16.x • WinCC RT Advanced V15.x • WinCC RT Advanced V14.x • WinCC RT Advanced V13.x • WinCC RT Advanced V12.x • WinCC RT Advanced V11.x
WinCC RT Professional V18.x	WinCC RT Professional (262144) WinCC RT Professional (153600) WinCC RT Professional (102400) WinCC RT Professional (65536) WinCC RT Professional (8192) WinCC RT Professional (4096) WinCC RT Professional (2048) WinCC RT Professional (512) WinCC RT Professional (128)	• WinCC RT Professional V18.x • WinCC RT Professional V17.x • WinCC RT Professional V16.x • WinCC RT Professional V15x • WinCC RT Professional V14.x • WinCC RT Professional V13.x • WinCC RT Professional V12.x • WinCC RT Professional V11.x
WinCC Unified PC V18.x	WinCC Unified PC (150) RT WinCC Unified PC (500) RT WinCC Unified PC (1k) RT WinCC Unified PC (2,5k) RT WinCC Unified PC (5k) RT WinCC Unified PC (10k) RT WinCC Unified PC (50k) RT WinCC Unified PC (100k) RT WinCC Unified PC (max.) RT	• WinCC Unified PC V18.x • WinCC Unified PC V17.x • WinCC Unified PC V16.x

Validity of licenses for WinCC options

WinCC options with version-independent license keys can also be used for future versions.

Starting without a valid license key

If you start the TIA Portal without a valid License Key , the system alerts you that you are working in non-licensed mode. You have the one-time option of activating a Trial License. However, this license is valid for a limited period only and expires after 21 days.

When the Trial License expires, the following scenarios can occur:

- TIA Portal has never been licensed on the PC in question:
  - No license-based actions can be performed in the TIA Portal.
- TIA Portal was already licensed on the PC in question:
  - A window requiring acknowledgment presents an alert for non-licensed mode every 10 minutes and for every action requiring a license.

License requirement for simulation In STEP 7

You do not require additional licenses when you use the menu command "Online > Simulation" to start the simulation in STEP 7.

If the following conditions are met, you need the appropriate licenses for the edition of STEP 7 that you have installed, even for the simulation:

- The connection to the PLC is configured and active.

License requirement for simulation In WinCC Runtime

If the following conditions are fulfilled, you also need the appropriate licenses for WinCC Runtime and for licensed options for the simulation:

- Start the simulator with the menu command "Online > Simulation".

If you do not have a valid license, you can activate the simulation for one hour.

Licenses for the TIA Portal Cloud Connector

For working with the TIA Portal Cloud Connector, you need a valid License Key for each device that you specify as "User device" in the TIA Portal Cloud Connector. The License Key is also required if the TIA Portal is installed on this device. A License Key is not required for devices that you are using as "remote device".

1.2.3 Licensing of WinCC Unified options

1.2.3.1 Logging

Logging

You have two options for logging your data in WinCC Unified:

- File-based logging
- Database-based logging

File-based logging

File-based logging allows you to log up to 5000 logging tags in an SQL Lite database.

Requirement

WinCC Unified Comfort Panel:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- No license is required for logging.

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for logging.

Licensing

To log up to a maximum of 5000 logging tags you need one or more licenses depending on the number of tags to be logged. To do this, refer to the licenses in the table below.

Upgrade

You do not need an upgrade license for logging when you upgrade the system.

Trial

If you do not have a valid license, you can configure the logging tags in trial mode.

Database-based logging

Database-based logging allows you to log all logging tags up to the high limit in an MS SQL database.

Requirement

WinCC Unified Comfort Panel:

- Not available

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for logging.

Licensing

Besides the functionality, database-based logging also includes an MS SQL server. Therefore, you need the "WinCC Unified Database Storage" license.

You also need one or more license(s) according to the number of tags to be logged. To do this, refer to the licenses in the table below.

File-based or database-based licenses

The license(s) required for the number of tags to be logged can be found in the table below:

License[1]	Number of logging tags	File-based	Database-based
WinCC Unified PC 100 Logging Tags	100	X	X
WinCC Unified PC 500 Logging Tags	500	X	X
WinCC Unified PC 1000 Logging Tags	1000	X	X
WinCC Unified PC 5000 Logging Tags	5000	X	X
WinCC Unified PC 10000 Logging Tags	10000	---	X

License[1]	Number of logging tags	File-based	Database-based
WinCC Unified PC 30000 Logging Tags	30000	---	X
WinCC Unified Database Storage[2]	---	Not required	X

[1] You can cumulate the logging tag licenses. This means that the sum of licenses corresponds to the number of tags that need to be logged.

[2] The license allows logging in the MS SQL database.

Upgrade

Depending on the MS SQL server, an upgrade license for WinCC Unified Database Storage may be required when upgrading the system.

Old licenses must be updated if required.

Trial

No trial mode is available for database-based logging.

1.2.3.2 Parameter sets

Parameter sets

Requirement

WinCC Unified Comfort Panel:

- You need a WinCC Unified Engineering System (ES) license for configuring.

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

Licensing

To use parameter sets on a Runtime PC, you need a license:

License	WinCC Unified PC	WinCC Unified Comfort Panel
WinCC Unified Parameter Control (PC)	X	Not required

Upgrade

The license is version dependent and requires the corresponding upgrade license for the upgrade.

Trial

If you do not have a valid license, you can configure the parameter sets in trial mode.

1.2.3.3 Process diagnostics

Process diagnostics

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

Licensing

To use process diagnostics on a Runtime PC, you need a license:

License	WinCC Unified PC	WinCC Unified Comfort Panel
WinCC Unified ProDiag	X	Not required

Trial

If you do not have a valid license, you can configure the process diagnostics in trial mode.

1.2.3.4 Client

Requirement

WinCC Unified Comfort Panel:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- A WinCC Unified Comfort Panel can be accessed by a maximum of 3 simultaneous remote access operations.
- The license for first access is free of charge.
- The use of remote access must be approved in the firmware.

Licensing

For remote access, you need a license depending on the number and type of accesses. The required license is listed in the table below:

	Monitoring only	Operator control and monitoring
WinCC Unified Panel Client Monitor (1)	X	---
WinCC Unified Panel Client Operate (1)	---	X

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

The following client functions are included in the license for WinCC Unified Runtime:

- Two accesses for operator control and monitoring (two local accesses or one local and one remote access)
- One access for monitoring (one local access or one remote access)

Licensing

For remote access, you need a license depending on the number and type of accesses. The required license is listed in the table below:

	Monitoring only	Operator control and monitoring
WinCC Unified Client Monitor (1)	X	---
WinCC Unified Client Monitor (3)	X	---
WinCC Unified Client Monitor (10)	X	---
WinCC Unified Client Monitor (30)	X	---
WinCC Unified Client Monitor (100)	X	---
WinCC Unified Client Operate (1)	---	X
WinCC Unified Client Operate (3)	---	X
WinCC Unified Client Operate (10)	---	X
WinCC Unified Client Operate (30)	---	X
WinCC Unified Client Operate (100)	---	X

Upgrade

The license is version independent and does not require any upgrade to a new version.

Trial

If you do not have a valid license, you can configure the client in trial mode.

1.2.3.5 Reporting

Reporting

Requirement

WinCC Unified Comfort Panel:

- You need a WinCC Unified Comfort (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

Licensing

You do not need a license to configure the Reporting option.

License

The option license "Configuration and manual Report Execution" is included in the WinCC Unified Engineering System and WinCC Unified RT licenses.

You need the "WinCC Unified Report Execution" license for automatic execution of Reporting (time, event).

License	WinCC Unified Comfort Panel	WinCC Unified PC
WinCC Unified Report Execution	X	X

Upgrade

The license is version dependent and requires the corresponding upgrade license for the upgrade.

Trial

If you do not have a valid license, you can configure the Reporting option in trial mode.

1.2.3.6 Openness

Openness

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

Licensing

No license is required for the Openness option.

Upgrade

No license is required for the upgrade.

Trial

If you do not have a valid license, you can configure the Openness option in trial mode.

1.2.3.7 Unified Collaboration

Unified Collaboration

Requirement

WinCC Unified Comfort Panel:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

Licensing

You do not need a license to configure the Unified Collaboration option.

You need a license for each device involved to use the Unified Collaboration option.

License	WinCC Unified Comfort Panel	WinCC Unified PC
WinCC Unified Collaboration	X	X

Upgrade

The license is version dependent and requires the corresponding upgrade license for the upgrade.

Trial

If you do not have a valid license, you can configure the Unified Collaboration option in trial mode.

1.2.3.8 Audit

Audit

Requirement

WinCC Unified Comfort Panel:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for use in runtime.

Licensing

You do not need a license to configure the Audit option.

You need a license to use the Audit option in runtime.

License

You can use the following licenses for the Audit option in WinCC Runtime:

License	Description
WinCC Unified Audit Basis	Secured communication Logging the Audit Trail Recording of process data changes (automatically, via script or system function) Confirmation with / without comment Exporting Audit Trail entries Audit Trail report and tamper indication Logging in and logging out Audit Trail entries Simple electronic signature Backing up and restoring Audit Trail database segments
WinCC Unified Audit Enhanced	In addition: Multiple electronic signature (2 persons) Display and analysis of the Audit Trail in the Audit Viewer Alias name for tag names

You need a license for each device involved to use the Audit option.

License	WinCC Unified Comfort Panel	WinCC Unified PC
WinCC Unified Audit Basis	X	X
WinCC Unified Audit Enhanced	X	X

Upgrade

The license is version dependent and requires the corresponding upgrade license for the upgrade.

Trial

If you do not have a valid license, you can configure the Audit option in trial mode.

1.2.4 Licensing of Plant Intelligence options

1.2.4.1 Calendar

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for the Calendar PI option in Runtime.

Licensing

You do not need an additional license to configure the Calendar PI option.

You need the following licenses to use the Calendar PI option in Runtime:

License	Description
WinCC Unified Calendar Base	Per server: • Three independent basic calendars • Unlimited number of derived calendars • Unrestricted use of the option • Version-based
WinCC Unified Calendar Extension (1)*	• One additional basic calendar

* Valid per extension, is independent of version and cumulative

Example:

WinCC Unified Calendar Base + WinCC Unified Calendar Extension (1) → 3 + 1 = 4

Upgrade

When upgrading to the next higher version, you need a WinCC Unified Calendar Base Upgrade license.

On the other hand, the WinCC Unified Calendar Extension (1) license is version-independent.

Trial

If you do not have a valid license, you can configure the Calendar PI option in trial mode.

1.2.4.2 Performance Insight

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for the Performance Insight PI option in Runtime.

Licensing

You do not need an additional license to configure the Performance Insight PI option.

You need the following licenses to use the Performance Insight PI option in Runtime:

License	Description
WinCC Unified Performance Insight Base	Per server: • Three analyzed plant objects • Unrestricted use of the option • Version-based
WinCC Unified Performance Insight Extension (10)*	• 10 additional analyzed plant objects
WinCC Unified Performance Insight Extension (30)*	• 30 additional analyzed plant objects

License	Description
WinCC Unified Performance Insight Extension (100)*	• 100 additional analyzed plant objects
WinCC Unified Performance Insight Extension (300)*	• 300 additional analyzed plant objects

* Valid per extension, is independent of version and cumulative

Example:

WinCC Unified Performance Insight Base + WinCC Unified Performance Insight Extension(10) → 3 + 10 = 13

WinCC Unified Performance Insight Base + WinCC Unified Performance Insight Extension (10) + WinCC Unified Performance Insight Extension (30) → 3 + 10 + 30 = 43

Upgrade

When upgrading to the next higher version you need a WinCC Unified Performance Insight Base Upgrade license.

On the other hand, the WinCC Unified Performance Insight Extension licenses are version-independent.

Trial

If you do not have a valid license, you can configure the Performance Insight PI option in trial mode.

1.2.4.3 Sequence

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for the Sequence PI option in Runtime.

Licensing

You do not need an additional license to configure the Sequence PI option.

You need the following licenses to use the Sequence PI option in Runtime:

License	Description
WinCC Unified Sequence Base	Per server: • 3 concurrent equipment modules • Unrestricted use of the option • Version-based
WinCC Unified Sequence Extension (5)*	• 5 additional concurrent equipment modules

* Valid per extension, is independent of version and cumulative

Example:

WinCC Unified Sequence Base + WinCC Unified Sequence Extension(5) → 3 + 5 = 8

WinCC Unified Sequence Base + WinCC Unified Sequence Extension (5) + WinCC Unified Sequence Extension (5) → 3 + 5 + 5 = 13

Upgrade

When upgrading to the next higher version you need a new WinCC Unified Sequence Base Upgrade license.

On the other hand, the WinCC Unified Sequence Extension (5) license is version-independent.

Trial

If you do not have a valid license, you can configure the Sequence PI option in trial mode.

1.2.4.4 Line Coordination

Requirement

WinCC Unified PC:

- You need a WinCC Unified Engineering System (ES) license for configuring.
- You need a WinCC Unified Runtime (RT) license for the Line Coordination PI option in Runtime.

Licensing

You do not need an additional license to configure the Line Coordination PI option.

You need the following licenses to use the Line Coordination PI option in Runtime:

License	Description
WinCC Unified Line Coordination Base	Per server: • 3 units • Unrestricted use of the option • Version-based
WinCC Unified Line Coordination Extension (5)*	• 5 units
WinCC Unified Line Coordination Extension (10)*	• 10 units
WinCC Unified Line Coordination Extension (50)*	• 50 units

* Valid per extension, is independent of version and cumulative

Example:

WinCC Unified Line Coordination Base + WinCC Unified Line Coordination Extension (5) → 3 + 5 = 8

WinCC Unified Line Coordination Base + WinCC Unified Line Coordination Extension (5) + WinCC Unified Line Coordination Extension (10)

→ 3 + 5 + 10 = 18

Upgrade

When upgrading to the next higher version you need a new WinCC Unified Line Coordination Base Upgrade license.

On the other hand, the WinCC Unified Line Coordination Extension licenses are version-independent.

Trial

If you do not have a valid license, you can configure the Line Coordination PI option in trial mode.

1.2.5 Handling licenses and license keys

Introduction

You need a valid License Key in each case to use STEP 7 Basic, STEP 7 Professional as well as WinCC Engineering System, options for WinCC Engineering System and WinCC Runtime.

NOTICE
Destruction of license keys by copying
A License Key cannot be copied. The copy protection prevents the License Keys from being copied. If there is an attempt to copy a License Key, the License Key gets destroyed.

Trial License

If you start the TIA Portal without a valid License Key, the system alerts you that you are working in non-licensed mode. You can activate a trial license once. The trial license expires after 21 days.

When the trial license has expired, the TIA Portal only runs with restrictions. For the full version of the TIA Portal, you need to purchase the corresponding license.

Installing license keys for STEP 7 and WinCC

Install the license with the Automation License Manager from the accompanying data storage medium to your PC.

You need additional License Keys to use WinCC Runtime or simulation on the Engineering-PC with the menu command "Online > Simulation > Start". You need to use Automation License Manager for this.

When you install a license, the relevant License Key will be removed from the original storage location of the License Keys.

Transferring license keys to the HMI device

You must transfer the License Keys to the HMI device to operate WinCC.

You transfer a License Key to the HMI device in the following cases:

- To operate WinCC Runtime
- To use add-ons for WinCC Runtime

When you transfer a license to an HMI device, the associated License Key is removed from the License Keys storage location.

If you no longer need the license or want to back up data, you must transfer the License Keys from the operator panel. You can then use this license on another PC or HMI device.

Uninstalling license keys

License Keys are always uninstalled using the Automation License Manager. Call the Automation License Manager before uninstalling

and back up the license key to be uninstalled to another storage location.

You uninstall a License Key in the following cases:

- When backing up data.
- If you no longer require the license.

You can also use a valid license on another PC.

Modification to the data media of the engineering system

Modifying the data medium of the engineering system can destroy the License Key. To do this before such modifications, open the Automation License Manager and back up License Key to be uninstalled to another storage location.

NOTICE
Destruction of license keys on PCs If one of the following cases applies, first, uninstall all License Keys : • Format the hard disk • Compress the hard disk • Restore the hard disk • Start an optimization program that moves fixed blocks • Install a new operating system Read the description of Automation License Manager ("Start > Siemens Automation > Documentation"). Comply with all warnings and notices.

Data backup of operating panels (Backup/Restore)

When backing up data on the HMI device, remove the License Keys on the HMI device. To do this, open the Automation License Manager and back up the License Key to another storage location.

NOTICE
Destruction of license keys on non-PC-based HMI devices License keys transferred as a result of backup/restore operations are destroyed in the case of the following HMI devices: • 270s series • 370s series Proceed as follows before restoring: 1. Use the Automation License Manager and ProSave to check if there are License Keys on the HMI device. 2. Transfer the existing License Keys from the HMI device. After restoring has been carried out, transfer the License Keys back to the HMI device.

Invalid license after time zone change

The installed licenses will no longer work if you change the time zone on a PC as follows: from a full-hour time zone to a time zone that is not based on a full hour.

To avoid this inconvenience, uninstall the License Key with the Automation License Manager under the time zone setting that was set when the License Keys was installed.

Example: You want to use the HMI device from the time zone "GMT +3:00" in the time zone "GMT +3:30".

1. Uninstall the License Key with the time zone setting "GMT +3:00".
2. Change the time zone settings accordingly on your PC.
3. Install the License Key.

This behavior does not apply to the Trial License.

Defective license

A license is defective in the following cases:

- If the License Key is no longer accessible at the storage area.
- If the License Key disappears during its transfer to the destination drive.

Note

If you reset the system date to an earlier time, all licenses are invalidated.

You can use the Automation License Manager to repair the defective license. Use the "Restore" function or the "Restore wizard" of the Automation License Manager

for this purpose. Contact Customer Support to restore them. You can find more detailed information on the Internet: Siemens Industry Online Support (https://support.industry.siemens.com/cs/de/en/)

Note

Runtime can also be operated without errors if the license is missing or defective. The system alerts you at brief intervals that you are working in non-licensed mode.

Microsoft SQL Server

A license is required for use of the Microsoft SQL Server database. The license is included in a properly licensed and installed version of WinCC.

The SQL Server which is licensed with the installation of WinCC can only be used in connection with WinCC.

Its use for other purposes requires an additional license. These include, for example:

- Use for custom databases
- Use for third-party applications
- Use of SQL access mechanisms which are not provided via WinCC.

Uninstalling

After uninstalling WinCC, you must uninstall the "WinCC" SQL server instance. To do this, select the entry "Microsoft SQL Server 20.." under "Control Panel > Programs" and uninstall it.

1.3 System requirements for installation

1.3.1 Notes on licenses

Availability of licenses

The licenses for the products of the TIA Portal are shipped on an installation data medium or via online software delivery (OSD).

Before you uninstall the TIA Portal, you must transfer and back up the licenses still required. Use the Automation License Manager for this purpose.

Provision of the Automation License Manager

The Automation License Manager is supplied on the installation data medium and is transferred automatically during the installation process.

If you uninstall the TIA Portal, the Automation License Manager remains installed on your system.

Working with the Automation License Manager

The Automation License Manager is a product of Siemens AG, which is used for handling license keys (technical representatives of licenses).

Software products that require license keys for operation, such as the TIA Portal, register the need for license keys automatically with the Automation License Manager . If the Automation License Manager finds a valid license key for this software, the software can be used according to the license usage terms associated with this license key.

Note

For additional information on how to manage your licenses with the Automation License Manager , refer to the documentation supplied with the Automation License Manager .

1.3.2 General software and hardware requirements

System requirements for installation

The following table shows the minimum software and hardware requirements that have to be met for the installation:

Hardware/software	Requirement
Processor	Intel® Core™ i3-6100U, 2.30 GHz
RAM	8 GB
Hard disk	S-ATA with at least 20 GB available space
Network	From 100 Mbit

Hardware/software	Requirement
Screen resolution	1024 x 768
Operating systems *	**Windows 10 (64-bit)** • Windows 10 Professional Version 21H1 • Windows 10 Professional Version 21H2 • Windows 10 Enterprise Version 2009/20H2 • Windows 10 Enterprise Version 21H1 • Windows 10 Enterprise Version 21H2 • Windows 10 Enterprise 2016 LTSB • Windows 10 Enterprise 2019 LTSC • Windows 10 Enterprise 2021 LTSC **Windows 11 (64-bit)** • Windows 11 Home Version 21H2 • Windows 11 Professional Version 21H2 • Windows 11 Enterprise 21H2 **Windows Server (64-bit)** • Windows Server 2016 Standard (full installation) • Windows Server 2019 Standard (complete installation) • Windows Server 2022 Standard (full installation)

* Including all security updates up to Microsoft Security Bulletin Summary for August 2022 (MS22-AUG). For more detailed information on operating systems, refer to the help on Microsoft Windows or the Microsoft Web site.

Some protocols might also support additional Windows versions. You can find more information in the product-specific requirements or check the compatibility with the compatibility tool. The compatibility tool is available on the Internet at https://support.industry.siemens.com/kompatool/pages/main/index.jsf?.

Recommended PC hardware

The following table shows the recommended hardware requirements:

Hardware/software	Requirement
Computer	SIMATIC FIELD PG M6 Comfort or higher (or comparable PC)
Processor	Intel® Core™ i5-8400H (2.5 to 4.2 GHz; 4 cores + Hyper-threading; 8 MB Smart Cache)
RAM	16 GB or more (32 GB for large projects)
Hard disk	SSD with at least 50 GB available memory space
Network	1 GBit (for multiuser)
Monitor	15.6" Full HD Display (1920 x 1080 or higher)

Supported virtualization platforms

You can install the software packages "SIMATIC STEP 7" and "SIMATIC WinCC" on a virtual machine. For this purpose, use one of the following virtualization platforms in the specified version or a newer version:

- VMware vSphere Hypervisor (ESXi) 6.7 or higher
- VMware Workstation 12.5.5 (only WinCC)
- VMware Workstation 15.5.0 or higher
- VMware Player 12.5.5 (WinCC only)
- VMware Player 15.5.0 or higher
- Microsoft Hyper-V Server 2019 or higher

The following host and guest operating systems are recommended for these virtualization platforms:

Operating system	VMware vSphere Hypervisor (ESXi)	VMware Workstation	VMware Player	Microsoft Hyper-V
Windows Server 2016 Standard (full installation)	G	-	-	G
Windows Server 2019 Standard (complete installation)	G	-	-	H/G
Windows Server 2022 Standard (full installation)	G	-	-	H/G
Windows 10 Enterprise Version 2009/20H2	H/G	H/G	H/G	G
Windows 10 Enterprise 2016 LTSB	H/G	H/G	H/G	G
Windows 10 Enterprise 2019 LTSC	H/G	H/G	H/G	G
Windows 10 Enterprise 2021 LTSC	H/G	H/G	H/G	G

H: Can be used as host operating system

G: Can be used as guest operating system

H/G: Can be used as host and guest operating system

-: Cannot be used as the host or guest operating system

Note

- The same hardware requirements apply to the host operating system as for the respective TIA products.
- The plant operator must ensure that sufficient system resources are available for the host operating systems.
- The hardware certified by the manufacturers is recommended for the use of HyperV server and ESXi.
- When you use Microsoft Hyper-V, accessible stations cannot be displayed.

Supported security programs

The following security programs are compatible with "SIMATIC STEP 7 Basic" and "SIMATIC WinCC":

- Antivirus programs:
  - Symantec Endpoint Protection 14.3
  - Trend Micro Office Scan 14.0
  - McAfee Endpoint Security (ENS) 10.6 and 10.7
  - Windows Defender
  - Qihoo 360 "Safe Guard 12.1" + "Virus Scanner"

  Note

  Make sure that your virus scanner and its databases are always up-to-date.

  The last update of the virus pattern was on 13 Sept. 2022.
- Encryption software:
  - Microsoft Bitlocker
- Host-based Intrusion Detection System:
  - McAfee Application Control 8.3.3

1.3.3 Product-specific special characteristics

1.3.3.1 Uninstalling WinCC Unified

Uninstalling WinCC

If you have already installed an update for WinCC Unified V16 and want to install WinCC V17, proceed as follows:

1. Open the Control Panel.
2. Start the uninstallation of SIMATIC WinCC Unified PC V16.0 UPDx.
3. When selecting the components, select "SIMATIC WinCC Update 16.0 UPDx" for uninstallation.
4. After completing this uninstallation, uninstall all desired components of WinCC V16 via the installation medium.

The information is contained in the readme under "Important notes (WinCC Unified)".

1.3.3.2 Installation of WinCC Runtime Unified

Specific requirements for the operating system and software configuration must be met for the installation.

Installation in domains and workgroups

WinCC Unified is generally approved for operation in a domain or workgroup.

However, be aware that domain group policies and restrictions of the domain might hinder the installation. In this case, remove the computer from the domain before installing WinCC Unified and Microsoft SQL Server. Log on to the local machine with administrator rights. Perform the installation. After successful installation, you can restore the WinCC computer to the domain. If the domain group policies and restrictions of the domain do not impede the installation, the computer need not be removed from the domain during the installation.

Be aware that domain group policies and restrictions of the domain might also hinder operation. If you cannot avoid these restrictions, run the WinCC computer in a workgroup.

Consult with the domain administrator if needed.

Operation on a network server

It is not permitted to operate WinCC Unified Runtime on a network server (e.g. domain controller, file server, name service server, router, software firewall, media server, exchange server).

Windows computer name

Before you start the WinCC installation, specify the Windows computer name. Follow the Windows naming rules.

Note

How to proceed after a subsequent change of the computer name is described in the WinCC Unified Runtime installation manual in the section "Changing the computer name or IP address".

The following characters are not permitted for the computer name:

- . , ; : ! ? " ' ^ ` ~ _
- + = / \ | @ * # $ % &
- ( ) [ ] { } < >
- Space

Follow these recommendations when assigning the Windows computer name:

- Only uppercase letters may be used.
- The first character must be a letter.
- The first 12 characters of the computer name must be unique.
- The computer name can have a maximum of 15 characters.

Hardware requirements for the installation

The following table shows the minimum hardware requirements that have to be met for the installation:

Hardware	Requirement
Processor type	Intel Core i3
RAM	4 GB
Free hard disk space	10 GB, 8 GB CF

Software requirements for the installation

Operating system

Software	Configuration	Comments
Windows 10 Pro	Windows 10 Pro Version 1909 (OS Build 18363) Windows 10 Pro Version 2004 (OS Build 19041) Windows 10 Pro Version 2009/20H2 (OS Build 19042) Windows 10 Pro Version 21H1 (OS Build 19043) Windows 10 Pro Version 21H2 (OS Build 19044)	64-bit
Windows 10 Enterprise	Windows 10 Enterprise Version 1909 (OS Build 18363) Windows 10 Enterprise Version 2004 (OS Build 19041) Windows 10 Enterprise Version 2009/20H2 (OS Build 19042) Windows 10 Enterprise Version 21H1 (OS Build 19043) Windows 10 Enterprise Version 21H2 (OS Build 19044)	
Windows 10 IoT Enterprise LTSB	Windows 10 Enterprise 2016 LTSB (OS Build 14393) (Test for IPC) Windows 10 Enterprise 2019 LTSC (OS Build 17763) (Test for IPC) Windows 10 Enterprise 2021 LTSC (OS Build 19044) (Test for IPC)	
Windows 11	Windows 11 Home Version 21H2 (OS Build 22000)	64-bit
Windows 11 Pro	Windows 11 Pro Version 21H2 (OS Build 22000)	
Windows 11 Enterprise	Windows 11 Pro Version 21H2 (OS Build 22000)	
Windows Server 2016 Standard Windows Server 2019 Standard Windows Server 2022 Standard	Full installation	64-bit

Note

Number of supported clients and connections

Desktop operating systems support a maximum of 5 clients. In server operating systems, more than 5 clients can connect to the server.

Windows limits the number of incoming connections on desktop operating systems to 20. This limits the number of possible accesses to runtime.

Compatible browsers

Operating system	Browser
Microsoft Windows	• Google Chrome • Microsoft Edge • Mozilla Firefox, Mozilla Firefox ESR
Android	• Google Chrome • Firefox • Edge
iOS, Mac	• Safari • Google Chrome • Firefox • Edge

More information on the use of browsers is available in the SIMATIC Unified PC Readme in the section "Internet browsers for WinCC Unified PC".

Windows specific software settings for IIS (Internet Information Services)

The following settings for IIS are automatically enabled in Windows during the installation of WinCC Runtime Unified:

- HTTP error
- HTTP Redirection
- Default document
- Static content
- .NET extensibility 3.5
- ASP
- ASP.NET 4.5
- ISAPI extensions
- ISAPI filters
- Dynamic Content Compression
- Static Content Compression
- Request Filtering

Table 1-1 Additional software requirements

Topic	Version / setting	Comment
Web browser	The browser must support HTML 5.	
User rights for installation	Administrator rights	
SOFTNET-IE S7 Lean Single License		You need this license to be able to operate Runtime Unified with up to 10 connections.
SIMATIC NET	V13 SP1	You need this license to be able to operate Runtime Unified with more than 10 connections.

Previously installed versions of WinCC Unified PC Runtime

The installation of Unified PC Runtime V18 is possible on devices for which the following applies:

- No Unified PC Runtime installed, or
- WinCC Unified PC Runtime V17 installed
  The V17 installation must not have been upgraded from V16.

Ports

When a Windows firewall is used, the installation routine of WinCC Unified Runtime sets up the following ports:

- HTTPS: 443
- Network Discovery (IS): 137
- Totally Integrated Automation administrator: 8888
- UMC AttachAgent: 4002
- OPC UA Discovery: 4840

If your system uses a different firewall solution, make sure the ports are set up accordingly.

You can find a list of the ports used by Unified Runtime in the user help SIMATIC Unified PC readme in the section "Security information".

Note

Disable HTTP port

For security reasons, it is recommended to disable port 80 on the IIS server:

1. On the Unified PC, launch "Internet Information Services (IIS) Manager".
2. Click "Default Web Sites" on the right.
3. Select "Remove" or "Manage Website > Exit".

Note

Blocked ports after operating system update or upgrade

Updating or upgrading the operating system of the Unified PC, e.g. from Windows Server 2016 to Windows Server 2019, may change the firewall settings. As a result, the OPC UA ports may be blocked.

If this happens, start the Siemens "Security Controller" tool and run "Restore settings".

Virtualization

You can install the "SIMATIC WinCC Runtime Unified" software package on a virtual machine. The following virtualization systems were tested:

- VMware vSphere Hypervisor (ESXi) 6.7 (or higher)
- VMware Workstation 12.5.5 and VMware Workstation 15.5.0 (or higher)
- VMware Player 12.5.5 and VMware Player 15.5.0 (or higher)
- Microsoft Hyper-V Server 2019 (or higher)

In a virtualization platform, all approved operating systems can be used as host operating system.

Requirement

The performance data of the virtual computers must meet the minimum requirements of WinCC clients.

Note

- Ensure that terminal and PLC networks are separated on the host system by using separate network adapters (dedicated, physically separated network adapters).
- The same hardware requirements apply to the host operating system as for the respective TIA products.
- The plant operator must ensure that sufficient system resources are available for the host operating systems.
- The hardware certified by the manufacturers is recommended for the use of HyperV server and ESXi.

Supported security programs

The following security programs are compatible with Unified Runtime:

Virus scanner	Symantec Endpoint Protection 14.3
	McAfee Endpoint Security (ENS) 10.6 and 10.7
	Trend Micro Office Scan 14.0
	Windows Defender (as part of the Windows operating system)
	Qihoo 360 "Safe Guard 12.1" + "Virus Scanner"

Whitelisting	McAfee Application Control 8.3.3
Hard disk encryption	Microsoft BitLocker (as part of the Windows operating system)

Principle

Care must be taken to ensure that the use of the antivirus software does not impair the process operation of a plant.

Rules for antivirus software (virus scanning clients)

- Integrated virus scanner firewall
  In WinCC Unified, the local Windows firewall used is configured with SIMATIC Security Control. Do not install or enable the integrated firewall of the antivirus software.
- Manual scan
  You must not perform a manual scan while Runtime is running. Perform disk on regular intervals on all plant PCs, for example, during the maintenance interval.
- Automatic scan
  For automatic scan it is sufficient to scan the incoming data traffic.
- Scheduled scan
  You must not perform a scheduled scan while Runtime is running.
- Pattern update
  The pattern update of the virus scanning clients (the plant PCs which are checked for viruses) performed by the higher-level virus scanning server (the plant PC which centrally manages the virus scanning clients).
- Dialog
  To avoid impairing the process operation, no dialog messages can be displayed on the virus scanning clients.
- Drives
  To prevent overlapping scans on network drives, only the local drives are scanned.

Otherwise apply the default settings.

What is secured?

The incoming data traffic is checked for viruses. The impairment of the process mode is minimized.

Note

If you are using an anti-virus scanner, make sure that the computer has sufficient system resources.

Supported database types

The following database types are supported by SIMATIC WinCC Unified PC:

HMI device	Supported database type
SIMATIC WinCC Unified PC	SQLite
	Microsoft SQL

Microsoft SQL for Unified PC

WinCC Unified PC uses SQLite as the default database type. To use Microsoft SQL, the system provides an installation option with a setup package.

- Logging with SQLite is not possible after the installation of Microsoft SQL.
- Existing SQLite files are retained, but they cannot be accessed in runtime.
- No backup can be created for SQLite.

Microsoft SQL Server 2017 is used as of TIA Portal V17. SQL Management Studio is no longer part of the Microsoft SQL Server installation package and is therefore not included in the Totally Integrated Automation Portal installation. You can install SQL Management Studio separately if needed.

To establish secure SQL Server connections, please observe the notes in the Microsoft documents:

- Server Network Configuration
- Enable encrypted connections to the Database Engine

1.4 Overview of processes and services of TIA Portal components

Additional processes and services are also installed during the installation of the TIA Portal or components of the TIA Portal.

The following tables provide an overview of these processes, the corresponding services and respective functions:

Table 1-2 Automation License Manager

Process	Corresponding service	Function
almsrv64x.exe	alm service	License management service / main process
almsrvbubble64x.exe	-	Process for tray icon info bubbles
almgui64x.exe	-	Graphical user interface of the Automation License Manager

Table 1-3 TIA administrator

Process	Corresponding service	Function
almsrv64x.exe	alm service	License management service
node.exe	SiemensTiaAdmin	Main process of the TIA Administrator website
TiaAdminNotifier.exe	-	User notifications and tray icon of the TIA Administrator

Table 1-4 ProSave

Process	Corresponding service	Function
PTProSave.exe	-	ProSave, graphical user interface main process
S7TraceService64x.exe	S7TraceServiceX	S7Dos related tracing
s7oiehsx64.exe	s7oiehsx64	S7Dos Helper Service
s7epasrv64x.exe	-	Event and parameter handling
s7oPNDiscoveryx64.exe	SIMATIC PnDiscovery Service	PN Discovery (accessibility via PN device name)
TraceConceptX.exe	TraceConceptX	S7Dos related tracing
ALMPanelPlugin.exe	-	Plugin for communication with Automation License Manager
-	S7DOS SCP Remote	Is also installed, but is only active with Cloud Connector
CommunicationSettings.exe	-	Setting access points

Table 1-5 WinCC Audit Viewer

Process	Corresponding service	Function
AuditViewer.exe	-	WinCC Audit Viewer main process

Table 1-6 Migration Tool

Process	Corresponding service	Function
Siemens.Automation.MigrationApplication.exe	-	Tool for converting WinCC and Simatic Manager projects into migration-compatible files

Table 1-7 Port Configuration Tool (PCT)

Process	Corresponding service	Function
Siemens.Simatic.Pct.ApplicationLoader.exe	-	Main process of the Port Configuration Tool (PCT)
s7oiehsx64.exe	s7oiehsx64	S7Dos Helper Service
s7epasrv64x.exe	-	Event and parameter handling
S7TraceService64x.exe	S7TraceServiceX	S7Dos related tracing
s7oPNDiscoveryx64.exe	SIMATIC PnDiscovery Service	PN Discovery (accessibility via PN device name)
TraceConceptX.exe	TraceConceptX	S7Dos related tracing
s7elonls64.exe	-	Routing between 32- and 64-bit application parts
CommunicationSettings.exe	-	Setting access points
-	S7DOS SCP Remote	Is also installed, but is only active with Cloud Connector

Table 1-8 Cloud Connector

Process	Corresponding service	Function
CloudConfigurator.exe	-	GUI for configuring the Cloud Connector
s7oiehsx64.exe	s7oiehsx64	S7Dos Helper Service
s7epasrv64x.exe	-	Event and parameter handling
S7TraceService64x.exe	S7TraceServiceX	S7Dos related tracing
s7oPNDiscoveryx64.exe	SIMATIC PnDiscovery Service	PN Discovery (accessibility via PN device name)
TraceConceptX.exe	TraceConceptX	S7Dos related tracing
CC.TunnelServiceHost.exe	S7DOS SCP Remote	Host of the tunnel service for the Cloud Connector
CommunicationSettings.exe	-	Setting access points

Table 1-9 Project server

Process	Corresponding service	Function
Siemens.Automation.Portal.Server(.exe)	V[Version number]prjsrv	TIA Portal project server main process
Siemens.Automation.Portal.Project.Server(.exe)	-	TIA Portal project server main process
Siemens.Automation.Portal.Server.Configuration(.exe)	-	User interface for configuring the project server
Siemens.Automation.Portal.Server.Administration(.exe)	-	User interface for the administration of the project server
Siemens.Automation.ProjectServer.AccessControl(.exe)	-	Controls the access authorization of the administration tool

Table 1-10 WinCC Runtime Advanced

Process	Corresponding service	Function
ScsServer.exe	-	Central data communication
StartCenter.exe	-	Start Center which permits starting of runtime, for example
HmiRTm.exe	-	The actual WinCC Advanced Runtime
s7elonls64.exe	-	Routing between 32- and 64-bit application parts
Miniweb.exe	-	Web contents of the WinCC Runtime Advanced
CodeMeterCC.exe	-	Runtime service of the code meter
CmWebAdmin.exe	CmWebAdmin.exe	Web administrator of the code meter
SmartServer.exe	cortsmartserver	Smart Server host process for remote connection
CodeMeter.exe	CodeMeter.exe	Code meter licensing during runtime
S7TraceService64x.exe	S7TraceServiceX	S7Dos related tracing
s7oiehsx64.exe	s7oiehsx64	S7Dos Helper Service

Process	Corresponding service	Function
s7epasrv64x.exe	-	Event and parameter handling
s7oPNDiscoveryx64.exe	SIMATIC PnDiscovery Service	PN Discovery (accessibility via PN device name)
-	OpcEnum	Service is only installed, but not started.

Table 1-11 TIA Portal with STEP 7 / WinCC Comfort / Advanced

Process	Corresponding service	Function
CodeMeterCC.exe	-	Runtime service of the code meter
Siemens.Automation.Portal.exe	-	The TIA Portal itself
Siemens.Automation.ObjectFrame.FileStorage.Server.exe	-	Process for data management in TIA Portal projects
Siemens.Automation.Diagnostics.CrashDetector.exe	-	Process for detecting TIA Portal crashes and displaying the crash box
Siemens.Automation.Tracing.ETW.EventCollector.ServiceHost.exe	Siemens Diagnostics Data Collector Service	Process for event data collection
s7oiehsx64.exe	s7oiehsx64	S7Dos Helper Service
s7epasrv64x.exe	-	Event and parameter handling
CmWebAdmin.exe	CmWebAdmin.exe	Web administrator of the code meter
CodeMeter.exe	CodeMeter.exe	Code meter licensing during runtime
S7TraceService64x.exe	S7TraceServiceX	S7Dos related tracing
Siemens.Simatic.TelemetryConnector.WindowsService.exe	Siemens Telemetry Connector Service	Service for collecting and sending telemetry data
s7oPNDiscoveryx64.exe	SIMATIC PnDiscovery Service	PN Discovery (accessibility via PN device name)
TraceConceptX.exe	TraceConceptX	S7Dos related tracing
IPCSecCom.exe	umscsvc	Communication to an UMC server
um.Ris.exe	-	Basic executable for UMC
um.sso.exe	-	Single sign-on for UMC
um.ess.exe	-	Basic executable for UMC
S7otbxsx64.exe	-	S7Dos block administration
-	OpcEnum	Service is only installed, but not started
-	UMC Service	Only used when UMC / UMAC is used

Table 1-12 STEP 7 / Safety / WinCC Professional ES incl. simulation

Process	Corresponding service	Function
CodeMeterCC.exe	-	Runtime service of the code meter
Siemens.Automation.Portal.exe	-	The TIA Portal itself
Siemens.Automation.ObjectFrame.FileStorage.Server.exe	-	Process for data management in TIA Portal projects
Siemens.Automation.Diagnostics.CrashDetector.exe	-	Process for detecting TIA Portal crashes and displaying the crash box

Process	Corresponding service	Function
Siemens.Automation.Tracing.ETW.EventCollector.ServiceHost.exe	Siemens Diagnostics Data Collector Service	Process for event data collection
s7oiehsx64.exe	s7oiehsx64	S7Dos Helper Service
s7epasrv64x.exe	-	Event and parameter handling
CmWebAdmin.exe	CmWebAdmin.exe	Web administrator of the code meter
CodeMeter.exe	CodeMeter.exe	Code meter licensing during runtime
S7TraceService64x.exe	S7TraceServiceX	S7Dos related tracing
Siemens.Simatic.TelemetryConnector.WindowsService.exe	Siemens Telemetry Connector Service	Service for collecting and sending telemetry data
s7oPNDiscoveryx64.exe	SIMATIC PnDiscovery Service	PN Discovery (accessibility via PN device name)
TraceConceptX.exe	TraceConceptX	S7Dos related tracing
IPCSecCom.exe	umscsvc	Communication to an UMC server
um.Ris.exe	-	Basic executable for UMC
um.sso.exe	-	Single sign-on for UMC
um.ess.exe	-	Basic executable for UMC
S7otbxsx64.exe	-	S7Dos block administration
CCDeltaLoader.exe	CCDeltaLoader	Responsible for determining whether a delta download is possible
RedundancyControl.exe	RedundancyControl	Responsible for the determination and management of the state of the redundant WinCC partner
CCTextServer.exe	CCTextServer	Server for runtime display of texts
CCPerfMon.exe	CCPerfMon	WinCC STOBS, monitoring of the system parameters and WinCC processes
SCSFsX.exe	SCSFsX	WinCC File Service between clients and servers
CCRtsLoader_x64.exe	CCRtsLoader	WinCC RT Tag management
CCSystemDiagnosticsHost.exe	CCSystemDiagnosticsHost	Host for displaying system diagnostics
CcAlgRtServer.exe	CCAlgRtServer	WinCC Alarm Logging Runtime Server
CCArchiveManager.exe	CCArchiveManagerService	Management of WinCC archives
CCRedundancyAgent.exe	CCRedundancyAgent Service	Archiving component for managing and reconciliation of the RT archive
CCTlgServer.exe	CCTlgServer	WinCC Tag Logging Server
CCUsrAcv.exe	CCUsrAcv	Responsible for managing user archives
CCProfileServer.exe	CCProfileServer	Saving RT persistence data of the WinCC Controls
sqlservr.exe	MSSQL$WINCC	SQL server for WinCC
CCAgent.exe	CCAgent	WinCC RT component for communication between WinCC stations
CCEServer_x64.exe	CCEServer	WinCC RT component for communication between WinCC stations
CCProjectMgr.exe	CCProjectMgr	WinCC Project Manager, handling of WinCC projects
SDiagRT.exe	-	System diagnostics data collector
gscrt.exe	-	WinCC Global Script Runtime

Process	Corresponding service	Function
PassDBRT.exe	-	Managing login data and passwords
CCUAImport.exe	-	WinCC RT User Archive component for importing TIA recipes
script.exe	-	WinCC RT C Scripting component
PdlRt.exe	-	Graphic runtime
SCSDistServiceX.exe	SCS Distribution Service	WinCC RT component for communication between WinCC stations
SCSMX.exe	SCSMonitor	WinCC RT component for communication between WinCC stations
sqlbrowser.exe	SQLBrowser	Microsoft SQL process (WinCC uses SQL server)
sqlwriter.exe	SQLWriter	Microsoft SQL process (WinCC uses SQL server)
CCDBUtils.exe	CCDBUtils	WinCC component for managing the WinCC SQL database
CCRemoteService.exe	CCRemoteService	WinCC component for remote management of the WinCC SQL database
sqlceip.exe	SQLTELEMETRY$WINCC	Microsoft SQL process (WinCC uses SQL server)
CCEClient_x64.exe	CCEClient	WinCC RT component for communication between WinCC stations
RedundancyState.exe	RedundancyState	Client-side monitoring of the WinCC server states
CCPackageMgr.exe	CCPackageMgr	WinCC components for managing the server packages
CCNSInfo2Provider.exe	CCNSInfo2Provider	WinCC component for browsing WinCC/ STEP 7 tags
Siemens.Simatic.Srm.RdpComp.Data.ContextMgrX.exe	-	SCADA RT Professional Compiler component
CCDmRuntimePersistence.exe	-	WinCC RT component for reconciliation of the internal tags in redundant systems
CCWriteArchiveServer.exe	-	Responsible for writing archive data in the SQL database (3 instances: TLG-fast / TLG-slow / ALG)
CCLicenseService.exe	CCLicenseService	Responsible for allocation and release of licenses
CCUCSurrogate.exe	-	WinCC RT Professional application for the system tray icon in the Windows taskbar
Simulation.exe	-	WinCC RT Professional tag simulation
CCConfigStudio.exe	-	WinCC Config Studio is an application for the WinCC RT Professional tag simulation
WinCCChnDiag.exe	-	WinCC channel diagnostics
-	CCTMTimeSyncServer	Is used in redundant servers to synchronize time stamps
-	CcUaDAS	WinCC OPC UA component
-	CCOpcUaImporter	WinCC OPC UA component

Process	Corresponding service	Function
-	CCAlgIAlarmDataCollector	WinCC RT Professional alarm system
-	SQLAgent$WINCC	Microsoft SQL process (WinCC uses SQL server)
-	OpcEnum	OPC Foundation component (WinCC OPC); Service is only installed, but not started
-	UMC Service	Only used when UMC / UMAC is used

Table 1-13 WinCC Advanced Runtime Simulation

Process	Corresponding service	Function
HmiRTm.exe	-	The actual WinCC Advanced Runtime
Miniweb.exe	-	Web contents of the WinCC Runtime Advanced
SmartServer.exe	cortsmartserver	Smart Server host process for remote connection
HmiRTmSim.exe	-	WinCC Advanced Tag Simulation

Table 1-14 WinCC Unified Engineering Package

Process	Corresponding service	Function
opcualds.exe	UALDS	OPC UA Local Discovery Service
mDNSResponder.exe	OPCF Bonjour Service	DNS name resolution
GfxWebBrowser.exe	-	Graphic rendering

Table 1-15 WinCC Unified Simulation

Process	Corresponding service	Function
WCCILS7pComDrv(.exe)	-	Process for communication with S7-1200 and S7-1500 controllers
RTILtraceTool(.exe)	TraceLogger_WinCC_Unified_PC	Unified Simulation related tracing
RTILtraceTool(.exe)	TraceProfiler_WinCC_Unified_PC	Unified Simulation related tracing
w3wp(.exe)	-	IIS Process Worker, host process for web pages
WCCILscs(.exe)	WCCILScsService	Siemens Communication Service, responsible for the internal data communication
WCCILpmon(.exe)	-	WinCC Process Monitoring
WCCILalg(.exe)	-	WinCC AlarmLogging, responsible for logging alarms
WCCILevent(.exe)	-	WinCC Event handling, responsible for handling events
GfxLicenseServer(.exe)	-	Responsible for granting and releasing licenses

Process	Corresponding service	Function
JobSchedulerHost(.exe)	-	Responsible for scheduling jobs
GfxRTS(.exe)	-	Preparation of the graphical runtime
WCCILSDAMgr(.exe)	-	System Diagnostics Manager, responsible for the system diagnostics information
OpennessManagerHost(.exe)	-	Host for the Openness Manager
WCCILdist(.exe)	-	WinCC Unified Simulation basic process
WCCILdata(.exe)	-	WinCC Unified data manager
WCCILals(.exe)	-	WinCC AlarmServer, responsible for receiving and acknowledging alarms
WCCILpaco(.exe)	-	WinCC Parameter Control
WCCILtlg(.exe)	-	WinCC TagLogging, responsible for logging tags
WCCILs7(.exe)	-	Process for communication with S7-300 and S7-400 controllers
umcd(.exe)	UmclService	EMS basic process
idp(.exe)	UmclIdpService	EMS basic process
-	EventLogger	Event Logging Service
-	UmclWebUMService	Web service for User Management
-	umscsvc	User Management Secure Communication Service
-	w3logsvc	IIS Logging Service
-	WMSVC	IIS Web Management Service

Table 1-16 Add-Ins

Process	Corresponding service	Function
Siemens.Automation.AddIn.Rollout.Service.exe	Siemens Add-In Rollout Service	Mass rollout of Corporate Add-Ins

1.5 Using Security Logging

1.5.1 Security Logging in the TIA Portal

Security Logging in the TIA Portal is a function to acquire, save and analyze security-relevant event and log data from the Engineering System, as well as from components of the automation environment.

The event and log data is stored locally in Windows systems. This data can be analyzed, saved and exported through the Windows Event Viewer.

In a further step, the event and log data can be transferred to SIEM systems (**S**ecurity **I**nformation and **E**vent **M**anagement). Thus, Security Logging makes it possible to centrally collect and analyze security-relevant events from different systems in the network, and

to react to threats. Security Logging is part of a bundle of measures recommended by international security standards and regulations for increasing security.

Security Logging is disabled by default in the TIA Portal. Administrators can initially activate Security Logging through a batch file. When Security Logging has been activated, the function can be deactivated and re-activated either by means of batch files or through the Windows Registry.

See also

Activating and deactivating Security Logging (Page 75)

1.5.2 Activating and deactivating Security Logging

The event channel for Security Logging is registered when the TIA Portal is installed on the computer; however, the function is disabled by default. You can activate and deactivate Security Logging in the Windows Registry by setting the corresponding value in the "AuditLogOn" key. The value "1" activates Security Logging while other values deactivate it. By default, Security Logging is deactivated in the TIA Portal.

The "AuditLogOn" key is not created at the time of installation of the TIA Portal. There are two batch files in the installation directory of the TIA Portal in the "bin" folder:

- The batch file "SecurityAuditLoggingEnable.bat" creates the key "AuditLogOn" and activates Security Logging.
- The batch file "SecurityAuditLoggingDisable.bat" deactivates Security Logging.

Security Logging is version-specific. The setting does not have any effect on other versions of the TIA Portal.

Requirements

- You have Windows administrator rights.
- To log user accounts, the project must be protected.

Activating Security Logging with a batch file

1. Navigate to the "bin" folder in the installation directory of the TIA Portal.
2. Run the batch file "SecurityAuditLoggingEnable.bat".

Deactivating Security Logging with a batch file

1. Navigate to the "bin" folder in the installation directory of the TIA Portal.
2. Run the batch file "SecurityAuditLoggingDisable.bat".

Activating and deactivating Security Logging in the Windows Registry

1. In Windows, open the Registry Editor.
2. Go to the key "HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\Automation\SecurityLogging\18.0\Settings\AuditLogOn".
3. To activate Security Logging, in the "LoggingOn" key, input the value "1".
   To deactivate Security Logging, in the "LoggingOn" key, delete or change the value "1".
4. Confirm your input with "OK".

See also

Security Logging in the TIA Portal (Page 74)

Overview of events (Page 76)

1.5.3 Overview of events

The following tables provide an overview of user actions, the relevant log entries, the event categories and event types:

In addition to the content of the events listed in the following tables, each event contains the following information:

- Version of the TIA Portal
- Name of the project
- Name of the logged-on user from the user administration (UMAC - User Management and Access Control)

Note

Name of the logged-on user

The value for the name of the logged-on user from the user administration (UMAC) is set in three different categories:

- Category 1: The text "No user logged in." appears in the case where user administration (UMAC) is either not enabled in the project or no user is logged on outside the project on a device under "Accessible devices".
- Category 2: The text "Background process" appears in the case where UMAC is enabled for the project but a safety-critical operation is triggered by a background process (e.g. when a process loads data into a device in Multiuser Commissioning in "asynchronous mode").
- Category 3: The name of the logged-on user from the user administration (UMAC) appears in the case where UMAC is enabled for the project and the event was written by the TIA Portal main process.

Overview of events

Table 1-17 User administration (UMAC - User Management and Access Control)

User action	Log entry	Event category	Type of event
Creating a new local project user	Local user "{target user}" added	Access control	Information
Deleting a local project user	Local user "{target user}" deleted	Access control	Information
Adding a global user	Global user "{target user}" added	Access control	Information
Deleting a global user	Global user "{target user}" removed	Access control	Information
Adding a global user group	User group "{target user}" added	Access control	Information
Deleting a global user group	User group "{target user}" removed	Access control	Information
Logging on to a protected project	Local / Global user "{target user}" with role(s) "{semicolon seperated list of roles}" logged in successfully Local / Global user "{target user}" login failed Anonymous user with role(s) "{semicolon seperated list of roles}" logged in successfully Anonymous user login failed For Auto Log On No DSSO Session : Global user login failed as no desktop session is available	Access control	Information, errors
Authorized user logs out of a protected project	Local / Global user "{target user}" logged out Anonymous user logged out	Access control	Information
Changing the user name of a local project user	User "{target user}" renamed to "{new target user name}"	Configuration	Information
Changing the password of a local project user or one's own password	Local / Global user "{target user}" password changed	Configuration	Information
Activating a local project user, a global user or a global group	Local / Global user / group "{target user}" activated	Configuration	Information
Deactivating a local project user, a global user or a global group	Local / Global user / group "{target user}" de-activated	Configuration	Information
Changing the properties of a user	User "{target user}" property '{property name}' changed to {new property value}	Configuration	Information
Creating a user-defined role in the TIA Portal	Custom role "{role name}" created	Configuration	Information

User action	Log entry	Event category	Type of event
Deleting a user-defined role in the TIA Portal	Custom role "{role name}" deleted	Configuration	Information
Renaming a user-defined role in the TIA Portal	Custom role "{old role name}" renamed to "{new role name}"	Configuration	Information
Changing the assignment of a function right to a user-defined role in the TIA Portal.	Custom role "{role name}" function rights assignment changed to "{semicolon seperated list of function rights}"	Configuration	Information
Changing the runtime timeout of a user-defined role in the TIA Portal	Custom role "{role name}" property 'Runtime timeout' changed to {new runtime timeout}	Configuration	Information
Assigning a role to a user or a user group	Local user / Global user / User group "{target user name}" roles assignment changed to "{semicolon seperated list of roles}"	Configuration	Information

Table 1-18 Changes to CPU data (offline)

User action	Log entry	Event category	Type of event
Checking project integrity	Data integrity check succeeded / failed. A {data integrity error type} Integrity corruption found in object {object id} in project {project name}	AuditLog	Supervision successful, error

Table 1-19 Loading to CPU

User action	Log entry	Event category	Type of event
Loading a hardware configuration to a CPU	Download of hardware configuration to {PLC name} with address {address}	Configuration	Supervision successful
Loading a standard user program to a CPU (Start alert)	Software download started to target PLC: {PLC name} with address {address} Operating mode during download: {RUN|STOP} Type of download: {Delta|Complete}	Configuration	Information

User action	Log entry	Event category	Type of event
Loading a standard user program into a CPU (list of blocks to be loaded or deleted, if necessary several times)	Software objects downloaded (+) / deleted (-): {+|-} {full qualified name of object} {+|-} {full qualified name of object} ...	Configuration	Information
Loading a standard user program into a CPU (completion message with software signature)	Successful: Software download succeeded Software signature after sownload {signature|"null"} Error: Software download aborted with error: {error message} • PLC was left in an inconsistent state → Download in STOP • PLC software contents was rolled back → Download in RUN	Configuration	Supervision successful, error
Loading a failsafe program to a CPU	Downloaded safety program with the collective F-signature {collective F-signature}.	Configuration	Supervision successful

Table 1-20 Loading the device as a new station

User action	Log entry	Event category	Type of event
Loading a configuration from a CPU	Upload of {PLC name} with address {address}.	Configuration	Supervision successful, error

Table 1-21 Changes to CPU data (online)

User action	Log entry	Event category	Type of event
Access to a CPU	Successful: Login to Plc {PLC name} with address {address} is successful. Error: Login to Plc {PLC name} with address {address} failed.	Access control	Supervision successful, error
Forcing variables via online access to a CPU	Forcejob installed on target CPU: {PLC name} with address {address} Forced variables (address : value): Adr1 : value1 Adr2 : value2 ... Forcejob on target PLC replaced: {PLC name} with address {address} Forced variables (address : value): Adr1 : value1 Adr2 : value2 ... Forcejob on target CPU stopped: {PLC name} with address {Address}	Configuration	Supervision successful
Changing the IP/network configuration of a CPU via online access	Change the network settings of {device name}({address}) to [IpAddress:{address to set}| SubnetMask:{subnet mask}| Router address:{router address}]	Configuration	Supervision successful, error
Changing the password for protecting confidential configuration data via online access	Set password for "Protection of PLC configuration data" of {PLC name}({address}) Delete password for "Protection of PLC configuration data" of {PLC name}({address})	Configuration	Supervision successful, error
Changing the operating state of a CPU via online access	Set {PLC name}({address}) to Run/Stop	Configuration	Supervision successful, error
Changing the system time of a CPU via online access	Change the system time of {device name}({address}) to {date time}	Configuration	Supervision successful, error

User action	Log entry	Event category	Type of event
Backing up/restoring a CPU via online access	Backup: Backup created from {PLC name} with address {address}. Restore: Online backup {backup name} restored to {PLC Name} with address {address}.	Configuration	Supervision successful
Initiating firmware update of a CPU via online access	Firmware update of {PLC name} ({address}) to version {fw version}	Configuration	Supervision successful, error
Reset CPU to factory settings	Reset to factory setting of {PLC name} ({address}). Settings: • Delete IP address • Delete master password • Format memory card	Configuration	Supervision successful, error
MRES via online access	Memory reset of {PLC name} {address}	Configuration	Supervision successful, error
Format memory card	Formatting the memory card of {PLC name}{address}	Configuration	Supervision successful, error

Table 1-22 Changing the access level

User action	Log entry	Event category	Type of event
Changing the access level	Protection access level is changed from "{previously selected access level}" to "{current selection access level}" for {PLC name}	Configuration	Supervision successful, error
Configuring the password for full access	Protection Access password is configured successfully / failed to configure for "Full Access" for {PLC name}	Configuration	Supervision successful, error
Changing the password for full access	Protection access password is changed successfully / failed to change for "Full Access" for {PLC name}	Configuration	Supervision successful, error
Configuring the password for full access including failsafe (no protection)	Protection access password is configured successfully / failed to configure for "Full access incl. fail safe (no protection)" for {PLC name}	Configuration	Supervision successful, error
Changing the password for full access including failsafe (no protection)	Protection access password is changed successfully / failed to change for "Full Access incl. fail safe (no protection)" for {PLC name}	Configuration	Supervision successful, error

User action	Log entry	Event category	Type of event
Configuring the password for read access	Protection access password is configured successfully / fail to change for "Read Access" for {PLC name}	Configuration	Supervision successful, error
Changing the password for read access	Protection access password is changed successfully / fail to change for "Read Access" for {PLC-name}	Configuration	Supervision successful, error
Configuring the password for HMI access	Protection access password is configured successfully / failed to configure for "HMI Access" for {PLC name}	Configuration	Supervision successful, error
Changing the password for HMI access	Protection access password is changed successfully / fail to change for "HMI Access" for {PLC name}	Configuration	Supervision successful, error

Table 1-23 Protection of PLC Configuration Data

User action	Log entry	Event category	Type of event
Deactivating protection of confidential CPU configuration data	"Protection of PLC configuration data" is disabled / failed to disable for {PLC name}	Configuration	Supervision successful, error
Activating protection of confidential CPU configuration data	"Protection of PLC configuration data" is enabled / failed to enable for {PLC name}	Configuration	Supervision successful, error
Configuring the password for protection of confidential CPU configuration data	Password for "Protection of PLC configuration data" is configured successfully / failed to configure for {PLC name}	Configuration	Supervision successful, error
Changing the password for protection of confidential CPU configuration data	Password for "Protection of PLC configuration data" is changed successfully / failed to change for {PLC name}	Configuration	Supervision successful, error
Resetting the password for protection of confidential CPU configuration data	Password for "Protection of PLC configuration data" is reset successfully / failed to reset for {PLC name}	Configuration	Supervision successful, error

See also

Security Logging in the TIA Portal (Page 74)
Displaying and managing events (Page 83)

1.5.4 Displaying and managing events

Events can be displayed and managed in Windows in the Event Viewer. In addition to this, functions are provided via Microsoft Windows to track events automatically.
To centrally collect, analyze and further process events from different systems in the network, events can be transmitted in a SIEM (**S**ecurity **I**nformation and **E**vent **M**anagement) system. Use the documentation of the provider for information on transferring events to a SIEM system.

Displaying events in the Event Viewer

1. In Windows, open the Event Viewer.
2. In the left-hand column, go to the node "Event Viewer (Local) > Application and Services Logs > SiemensAG > Automation > TIAPortal".
3. Select the "Operational" log.
   The events of the log are displayed in the upper part of the main window.
4. Select an event.
   Information on the selected event is displayed in the lower part of the main window, in the tabs "General" and "Details".
5. To open the information on an event in a new window, double-click the event.

Managing events in the Event Viewer

1. In the Event Viewer, switch to the right-hand column, "Actions".
2. Use the actions to manage your log files.
   You can, for example, filter, search, save and delete logs.

Automated event tracking

The following Microsoft Windows functions can be used for automated event tracking:

- The command line program **wevtutil** can be used to read events from the command line. The program is provided via Microsoft Windows.
- TIA Portal provides the events via **E**vent **T**racing for **W**indows - **ETW**. Applications (for example, a SIEM system) that wish to call the events can access these events via the event source "SiemensAG-Automation-TIAPortal".

For more information on implementing automated event tracing, use Microsoft's documentation of features.

See also

Security Logging in the TIA Portal (Page 74)

1.6 Installation log

Function of the installation log

The progress during the following installation processes is logged in a file:

- Installing products
- Modifying or updating already installed products
- Repairing an existing installation
- Uninstalling products

If errors occur during the installation process or warnings are issued, these can be evaluated with the help of the log file. You can do this yourself or contact product support.

Installation logs storage location

The log file is the most recent file with the file extension ".log" and the name of which that starts with "SIA".

The location of the log file is stored in the environment variable "%autinstlog%". You can enter this environment variable in the address bar of Windows Explorer to open the folder with the log files. Alternatively, you can navigate to the corresponding directory by entering "CD %autinstlog%" in the command line.

The storage location is dependent on the operating system, e.g. "C:\ProgramData\Siemens\Automation\Logfiles\Setup" in English-language Windows.

Setup_Report (CAB file)

To make it easier to provide Product Support with all necessary files, an archive file that contains the installation log and all other required files is saved in CAB format. This log can be found at "%autinstlog%\Reports\Setup_report.cab". Send this CAB file to Product Support if you need assistance with installation. With this information, Product Support can determine whether the installation was executed properly. CAB files that were generated during earlier installation processes are saved with a date ID in the "Reports" directory.

1.7 Starting installation

Introduction

Software packages are installed automatically by the setup program. The setup program starts once the installation medium has been inserted in the drive.

Requirement

- Hardware and software of the programming device or PC meet the system requirements.
- You have administrator privileges on your computer.
- All running programs are closed.

Procedure

To install the software packages, follow these steps:

1. Insert the installation medium in the relevant drive.
   The setup program starts automatically unless you have disabled Autostart on the programming device or PC.
2. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
   The dialog for selecting the setup language opens.
3. Choose the language in which you want the setup program dialogs to be displayed.
4. To read the information on the product and installation, click "Read Notes" or "Installation Notes".
   The help file containing the notes opens.
5. Once you have read the notes, close the help file and click "Next".
   The dialog for selecting the product languages opens.
6. Select the languages for the product user interface, and click "Next".

   Note

   "English" is always installed as the basic product language.

   The dialog for selecting the product configuration opens.
7. Select the products you want to install:
   - If you wish to install the program in a minimal configuration, click "Minimal".
   - If you wish to install the program in a typical configuration, click "Typical".
   - If you wish to personally select the products to be installed, click "User-defined". Then select the check boxes for the products you wish to install.
8. If you want to create a shortcut on the desktop, select the "Create desktop shortcut" check box.
9. Click "Browse" if you want to change the target directory for the installation.

   Note

   Please note the following:
   - The length of the installation path must not exceed 89 characters.
   - You can only change the installation path if no other product from the software package which you intend to install, has been installed, yet.
   - To increase security, select a directory that is protected by administrative rights as the target directory.

10. Click "Next".
    The dialog for the license terms opens.
11. To continue the installation, read and accept all license agreements and click "Next".
    If changes to the security and permission settings are required in order to install the TIA Portal, the security settings dialog opens.
12. To continue the installation, accept the changes to the security and permissions settings, and click "Next".
    The next dialog displays an overview of the installation settings.
13. Check the selected installation settings. If you want to make any changes, click "Back" until you reach the point in the dialog where you want to make changes. Once you have completed the desired changes, return to the overview by clicking on "Next".
14. Click "Install".
    Installation is started.

    Note

    If no license key is found during installation, you have the chance to transfer it to your PC. If you skip the license transfer, you can register it later with the Automation License Manager.

    Following installation, you will receive a message indicating whether the installation was successful.

15. It may be necessary to restart the computer. If this is the case, select the "Yes, restart my computer now.". Then click "Restart".
16. If the computer does not reboot, click "Exit".

Result

The TIA Portal along with the products and licenses you have ordered and the Automation License Manager have been installed on your computer.

1.8 Displaying the installed software

You can find out which software is installed at any time. In addition, you can display more information on the installed software.

Procedure

To display an overview of the software installed, follow these steps:

1. Click "Installed software" in the "Help" menu.
   The "Installed software" dialog opens. You will see the installed software products in the dialog. Expand the entries to see which version is installed in each case.
2. If you would like to display additional information on the installed automation software, click the link on the "Detailed information about installed software" dialog.
   The "Detailed information" dialog opens.
3. Chose the topic you want more information about in the area navigation.

1.9 Modifying or updating installed products

You have the option to modify installed products using the setup program or to update to a new version.

Blocks with know-how protection from earlier versions of the TIA Portal are not automatically upgraded with the project. Remove the know-how protection of the blocks before you update the TIA Portal. Then set up the know-how protection with the current version of TIA Portal. For more detailed information, refer to the information system.

Requirement

- Hardware and software of the programming device or PC meet the system requirements.
- You have administrator privileges on your computer.
- All running programs are closed.

Procedure

To modify or update installed products, follow these steps:

1. Insert the installation medium in the relevant drive.
   The setup program starts automatically unless you have disabled Autostart on the programming device or PC.
2. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
   The dialog for selecting the setup language opens.
3. Choose the language in which you want the setup program dialogs to be displayed.
4. To read the information on the product and installation, click the "Read Notes" or "Installation Notes" button.
   The help file containing the notes opens.
5. Once you have read the notes, close the help file and click the "Next" button.
   The dialog for selecting the installation variant opens.
6. Select the "Modify/Upgrade" option button and click the "Next" button.
   The dialog for selecting the product languages opens.
7. Select the check boxes of the product languages that you want to install. You can remove previously installed product languages by clearing the corresponding check boxes.

   Note

   Note that the product language "English" cannot be removed.

8. Click the "Next" button.
   The dialog for selecting the product configuration opens.
9. Select the check boxes of the components that you want to install. You can remove previously installed components by clearing the corresponding check boxes.

10. Click the "Next" button.

    Note

    Note that you cannot change the target directory because the existing installation is being modified.

    If changes to the security and permission settings are required in order to install the TIA Portal, the security settings dialog opens.

11. To continue the installation, accept the changes to the security and permissions settings, and click the "Next" button.
    The next dialog displays an overview of the installation settings.

12. Click the "Modify" button.
    This starts the installation of the additional components.

    Note

    Following installation, you will receive a message indicating whether the existing installation was successfully changed.

13. It may be necessary to restart the computer. If this is the case, select the "Yes, restart my computer now." option button. Then click "Restart".

14. If the computer does not reboot, click "Exit".

Result

The existing installation has been modified on your computer.

1.10 Repairing installed products

You have the option to repair installed products by completely reinstalling them using the setup program.

Requirement

- Hardware and software of the programming device or PC meet the system requirements.
- You have administrator privileges on your computer.
- All running programs are closed.

Procedure

To repair installed products, follow these steps:

1. Insert the installation medium in the relevant drive.
   The setup program starts automatically unless you have disabled Autostart on the programming device or PC.
2. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
   The dialog for selecting the setup language opens.
3. Choose the language in which you want the setup program dialogs to be displayed.
4. To read the information on the product and installation, click the "Read Notes" or "Installation Notes" button.
   The help file containing the notes opens.
5. Once you have read the notes, close the help file and click the "Next" button.
   The dialog for selecting the installation variant opens.
6. Select the "Repair" option button, and click the "Next" button.
   The next dialog displays an overview of the installation settings.
7. Click the "Repair" button.
   This starts the repair of the existing installation.

   Note

   Following installation, you will receive a message indicating whether the existing installation was successfully repaired.

8. It may be necessary to restart the computer. If this is the case, select the "Yes, restart my computer now." option button. Then click "Restart".
9. If the computer does not reboot, click "Exit".

Result

The installed products have been reinstalled.

1.11 Starting to uninstall

Introduction

Software packages are removed automatically by the setup program. Once started, the setup program guides you step-by-step through the entire removal procedure.

You have two options for removing:

- Removing selected components via the Control Panel
- Removing a product using the installation medium

Note

The Automation License Manager will not be removed automatically when you remove the software packages, because it is used for the administration of several license keys for products supplied by Siemens AG.

Removing selected components via the Control Panel

To remove selected software packages, follow these steps:

1. Open the Control Panel.
2. Click "Programs and Features".
   A dialog with the list of installed programs opens.
3. Select the software package to be removed and click the "Uninstall" button.
   The dialog for selecting the setup language opens.
4. Select the language in which you want the dialogs of the setup program to be displayed and click "Next".
   The dialog for selecting the products you want to remove opens.
5. Select the check boxes for the products that you want to remove and click "Next".
   The next dialog displays an overview of the installation settings.
6. Check the list with the products to be removed. If you want to make any changes, click the "Back" button.
7. Click the "Uninstall" button.
   Removal begins.
8. It might be necessary to restart the computer. If this is the case, select the "Yes, restart my computer now." option button. Then click "Restart".
9. If the computer does not reboot, click "Exit".

Removing a product using the installation medium

To remove all software packages, follow these steps:

1. Insert the installation medium in the relevant drive.
   The setup program starts automatically unless you have disabled Autostart on the programming device or PC.
2. If the setup program does not start up automatically, start it manually by double-clicking the "Start.exe" file.
   The dialog for selecting the setup language opens.
3. Select the language in which you want the setup program dialogs to be displayed.
4. To read the information on the product and installation, click "Read product information" or "Read installation notes".
   The help file containing the notes opens.
5. Once you have read the notes, close the help file and click the "Next" button.
   The dialog for selecting the installation variant opens.

6. Select the "Uninstall" option button and click the "Next" button.
   The next dialog displays an overview of the installation settings.
7. Click the "Uninstall" button.
   Removal begins.
8. It might be necessary to restart the computer. If this is the case, select the "Yes, restart my computer now." option button. Then click "Restart".
9. If the computer does not reboot, click "Exit".

1.12 Installing updates and support packages

1.12.1 Checking availability of updates and support packages and installing them

By default, the TIA Portal checks automatically if new software updates or support packages are available, for example, Hardware Support Packages (HSPs). The automatic search for updates takes place after each computer restart and then cyclically every 24 hours. You can also deactivate the automatic search at any time or reactivate it. You can also search for updates manually.

If updates are found, you can download and install them.

Note

Updates and support packages from TIA Portal V13 or higher are supported.

Deactivate or activate automatic search for software updates

To deactivate or reactivate the automatic search for software updates, follow these steps:

1. Select the "Settings" command in the "Options" menu.
   The "Settings" window is displayed in the work area.
2. Select the "General > Software Updates" group in the area navigation.
3. Deselect the "Check for updates daily" check box if you want to deactivate the automatic search for software updates.
4. Select the "Check for updates daily" check box if you want to reactivate the automatic search for software updates.

Manually searching for software updates

If you want to search for software updates manually, follow these steps:

1. Click "Installed software" in the "Help" menu.
   The "Installed software" dialog opens.
2. Click "Check for updates".
   The TIA Administrator opens.
3. Click on the "Manage software" tile.
   The available updates are displayed.

Or:

1. Select the "Settings" command in the "Options" menu.
   The "Settings" window is displayed in the work area.
2. Select the "General > Software Updates" group in the area navigation.
3. Click "Check for updates now".
   The TIA Administrator opens.
4. Click on the "Manage software" tile.
   The available updates are displayed.

Or:

1. Open the TIA Administrator via "Start > All Programs > Siemens Automation > TIA Administrator".
2. Click on the "Manage software" tile and on the "Check for updates" button there.
   The available updates are displayed.

Setting the server

You must set the corresponding server depending on whether you want to download updates and/or support packages from the TIA Automation Update Server or from a corporate server. To do this, follow these steps:

1. Open the TIA Administrator.
2. Click "Options" and select the "TIA Automation Software Update Server" option in the dialog that appears in the "Server used to check for updates" area.
   The software searches for available updates on the server of the manufacturer.

Or:

1. Open the TIA Administrator.
2. Click "Settings" and select the "Corporate server" option in the "Software management" tab.

3. Enter the server URL which you received from your administrator. The URL must be entered either in the format https://[URL of the server]/[path to the file directory] or in the format https://[server IP address]/[path to the directory]. If you are not sure which format to use, contact your administrator!
   The software looks for available updates on the server of your company.
4. Check the name of the production line and change this if necessary "ProductionLine1" is set by default.
   The purpose of production lines is to provide different users with specific updates/support packages. If your company does not work with different production lines, retain the specified entry.
   You can find detailed information on creating projects and working with production lines in the help on the TIA Updater Corporate Configuration Tool.

You can set a different server on which the updates are to be searched for at any time. However, changing this setting is blocked during a download process. After switching the server, all downloaded updates and support packages are displayed, even if they are not available on the currently set server.

Installing updates/support packages

For the exact procedure for installing updates/support packages, refer to the TIA Administrator help.

Alternative procedures for the installation of support packages

Another procedure is available for the installation of a support package. To do this, follow these steps:

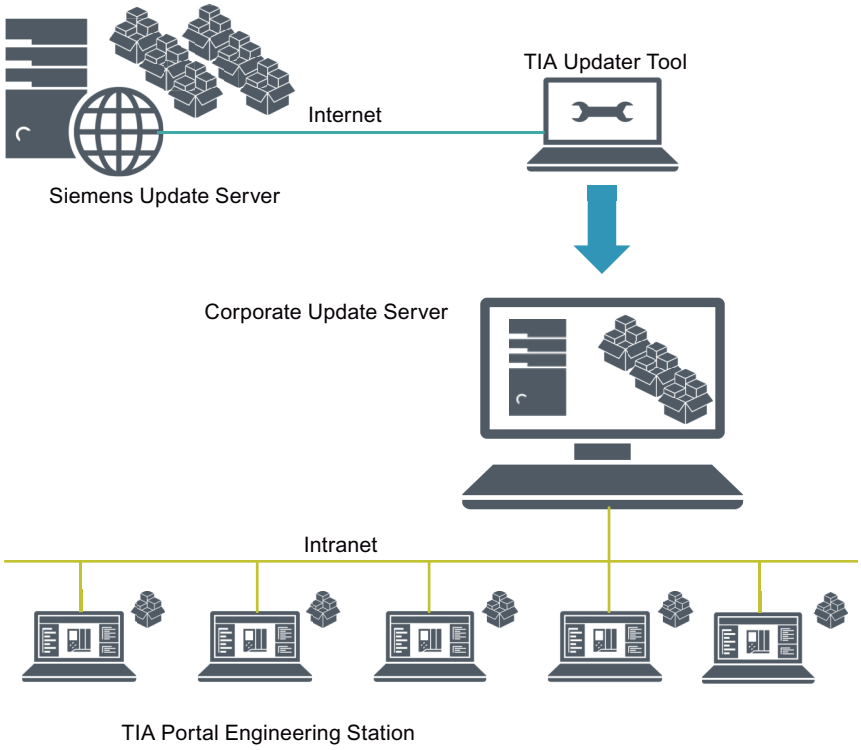
1. Click "Support packages" in the "Options" menu of the TIA Portal.
   The "Detailed information" dialog opens. A table lists all support packages from the directory that you selected as the storage location for support packages in the settings.
2. If you want to install a support package that is not in the list, you have the following options:
   - If the support package is already on your computer, you can add it to the list by selecting "Add from file system".
   - If you add a support package from the "Service & Support" page on the Internet, first you download it by selecting "Download from the Internet". Then you can add it from the file system.
3. Select the support package that you want to install.
4. Click "Install".
5. Close and then restart the TIA Portal.

1.12.2 Working with a company-internal server

1.12.2.1 Properties and advantage of a corporate server

Introduction

Using a corporate server, you can place selected updates / support packages on a local server and make them available to users, e.g. for different production lines. This has the advantage that users do not have to access the Internet, but can install updates via the intranet, an external hard disk, etc. Since users do not need direct access to the Internet, the protection against trojans or malicious software that contacts the Internet from the internal company network is significantly increased.



Configuring the server and creating projects

In the first phase, the corporate server is configured by a server administrator and the updates / support packages are deployed using the TIA Updater Corporate Configuration Tool. In addition, projects for different production lines can be created through which users can receive the updates they require. Users need to have access to the server area and have the name of the production line and be informed about the storage directory.

You can find detailed information on creating projects and working with production lines in the help on the TIA Updater Corporate Configuration Tool.

Working with the server

With the TIA Updater, users can download and install the updates and support packages that are relevant and relevant for them. Multiple download operations can be initiated at the same time. The installation of the updates / support packages must be done one after the other.

1.12.2.2 Configuring a corporate server for updates

Introduction

To provide available updates, support packages and language packages to users from a central location, you must configure a corporate server. Use the Microsoft Server Manager to do this.

The following is an example of the steps required to create and configure the server using the TIA Updater Corporate Configuration Tool (with the Microsoft Server 2016 operating system). Further settings which may be required for operation in your company are **not** covered in this description!

Note

Please note that the structure of the start menu and the storage location of the programs may vary depending on the different operating systems.

For more detailed information on configuration and operating the Server Manager, refer to the Microsoft help.

Requirement

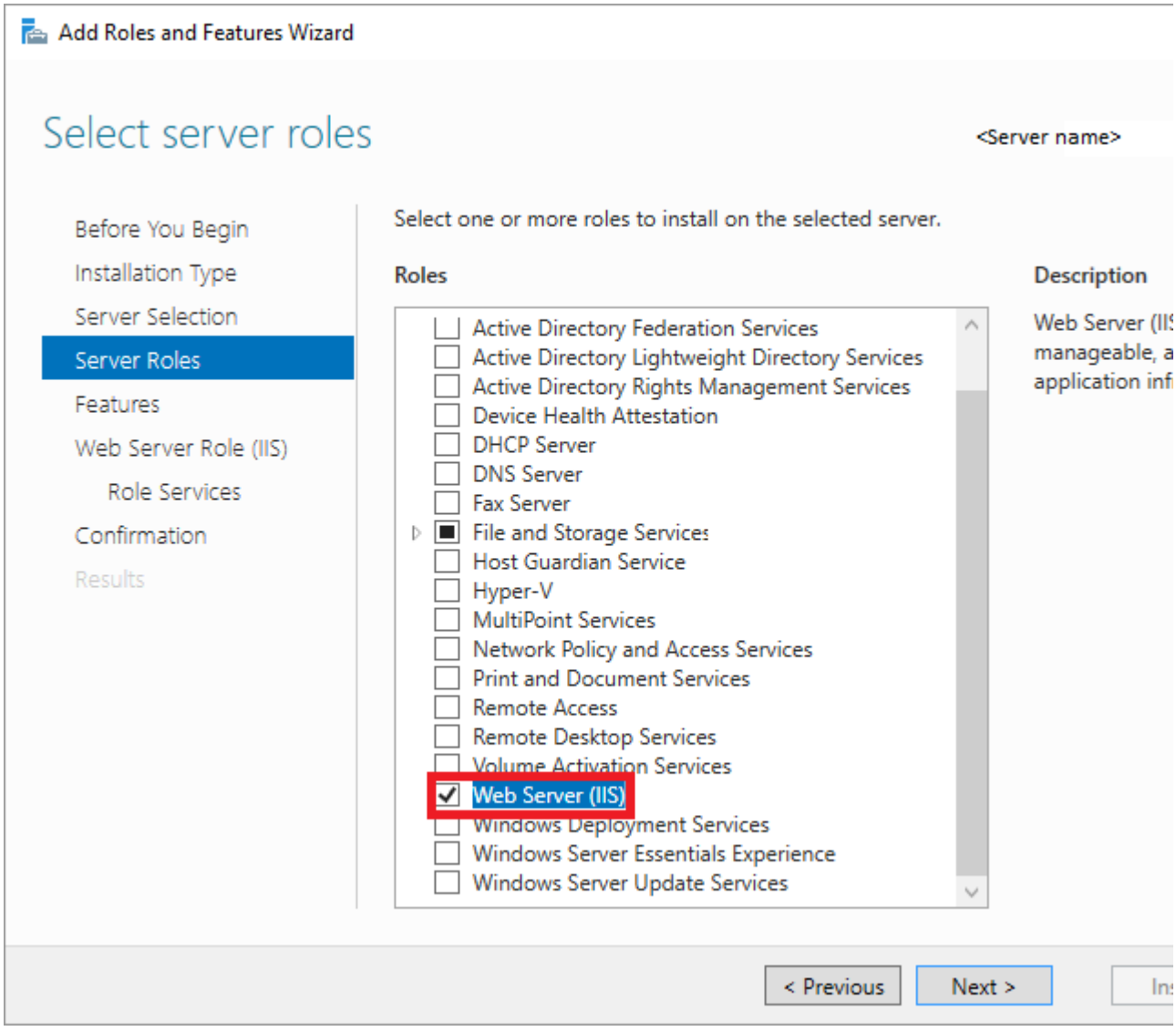
You must have administrator rights.

Install Web server role (IIS)

To install the required Web server role, follow these steps:

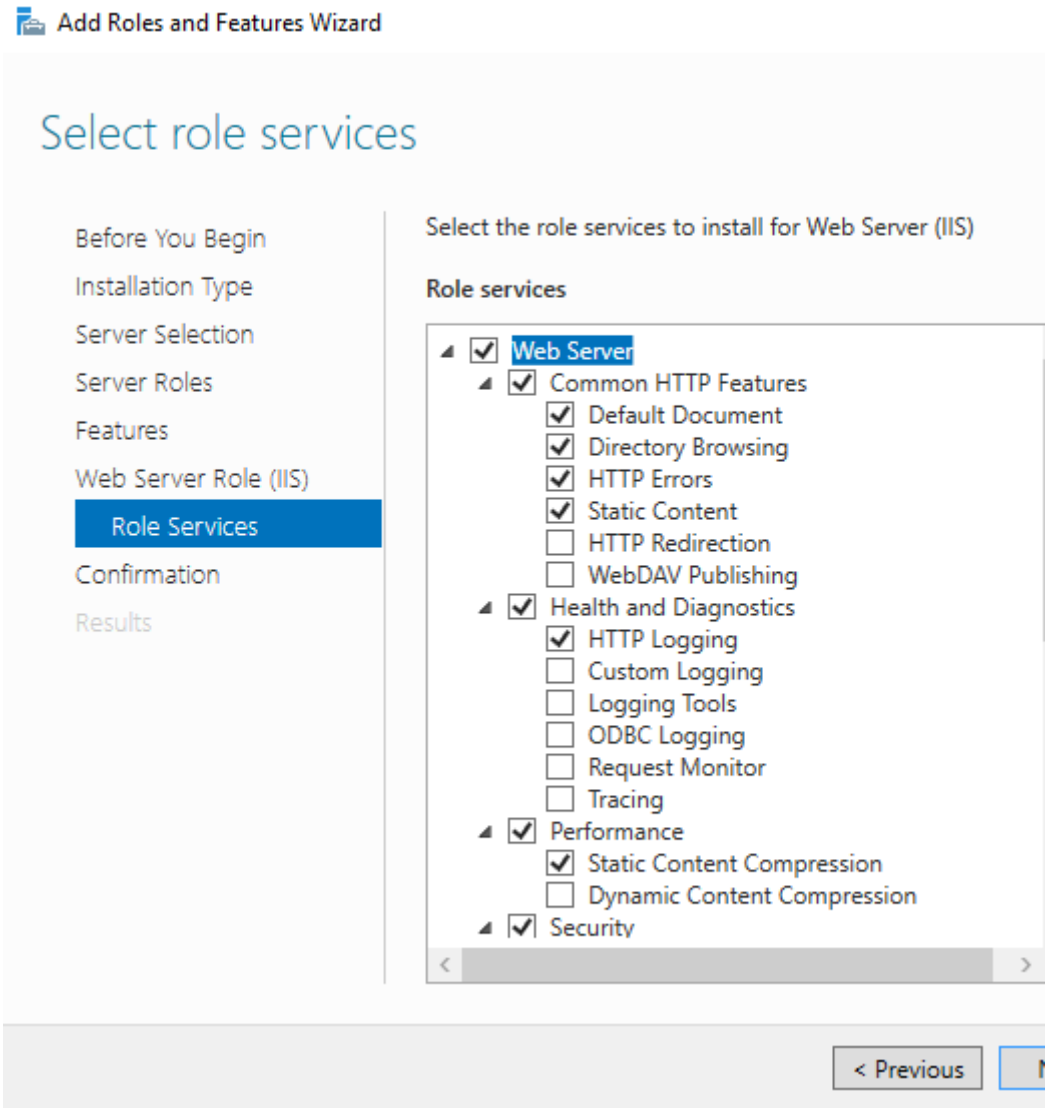
1. Open the start menu and select "Server Manager".
2. Click "Add roles and features" in the dashboard.
3. Use the "Add roles and features" wizard to add the Web server roles.
   Before the wizard starts, it is verified that a complex password has been assigned to the administrator, that the network settings have been configured accordingly and that the latest Windows security updates have been installed.
4. Select "Role-based or feature-based installation" as the installation type and click "Next".
5. Select the target server and click "Next".

6. Note the pre-selected roles that are installed by default, and then select the additional role "Web Server (IIS)".



7. Click "Add features" followed by "Next". The features for the web server are displayed.

8. Click "Role Services" and make sure that the following features are selected or select them:



"Web server" area

- Default document
- Directory browsing
- HTTP errors
- Static content

"Health and diagnostics" area

- HTTP logging
- Request monitor

"Performance features" area

- Static content compression

"Security" area

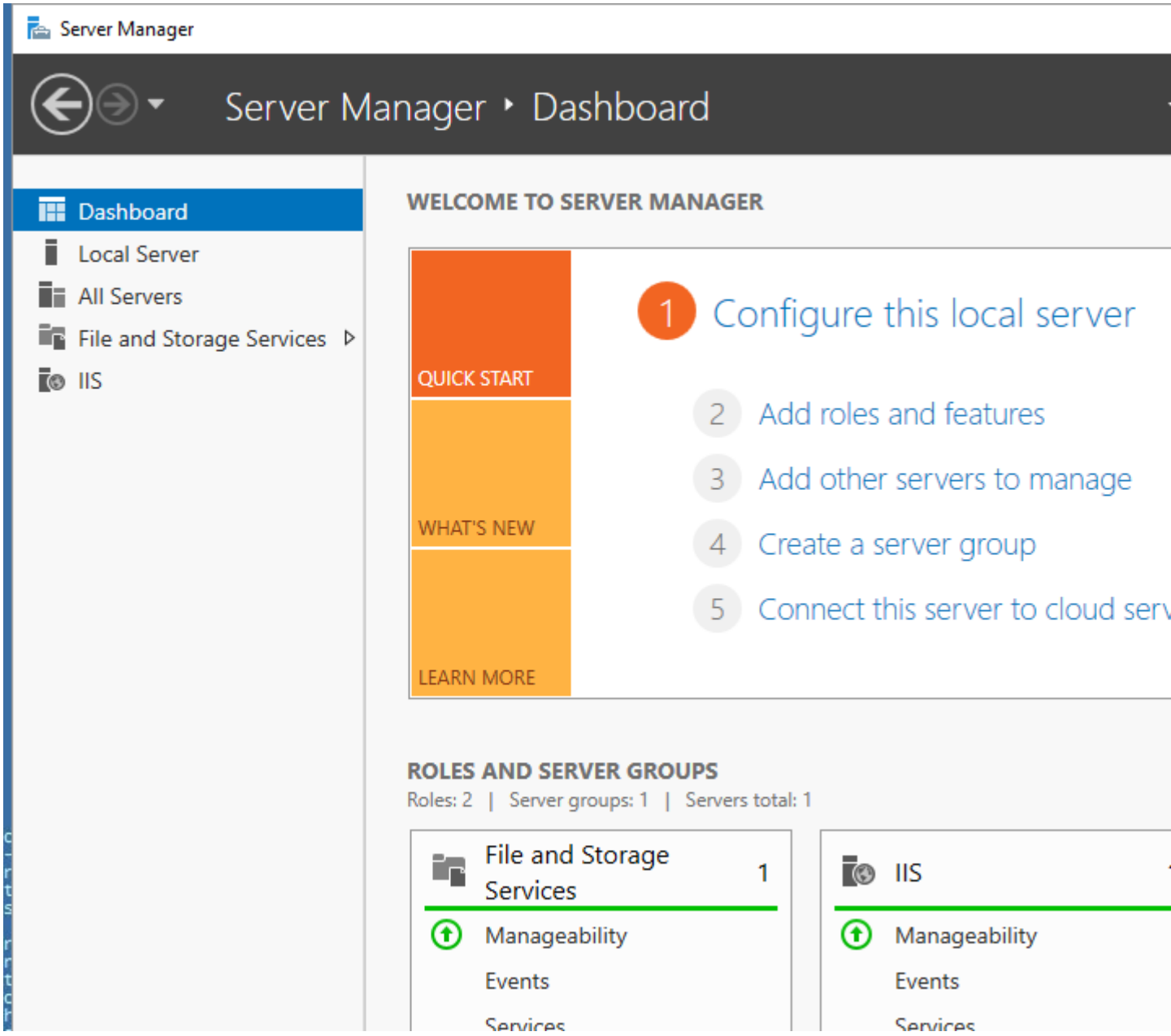
- Request filtering

"Application Development Features" area

- .NET Extensibility 3.5
- .NET Extensibility 4.6
- ASP.NET 3.5
- ASP.NET 4.6
- ISAPI Extensions
- ISAPI Filters

9. Click "Next".
10. Check your selection in the "Confirm installation selection" dialog box and click "Install".

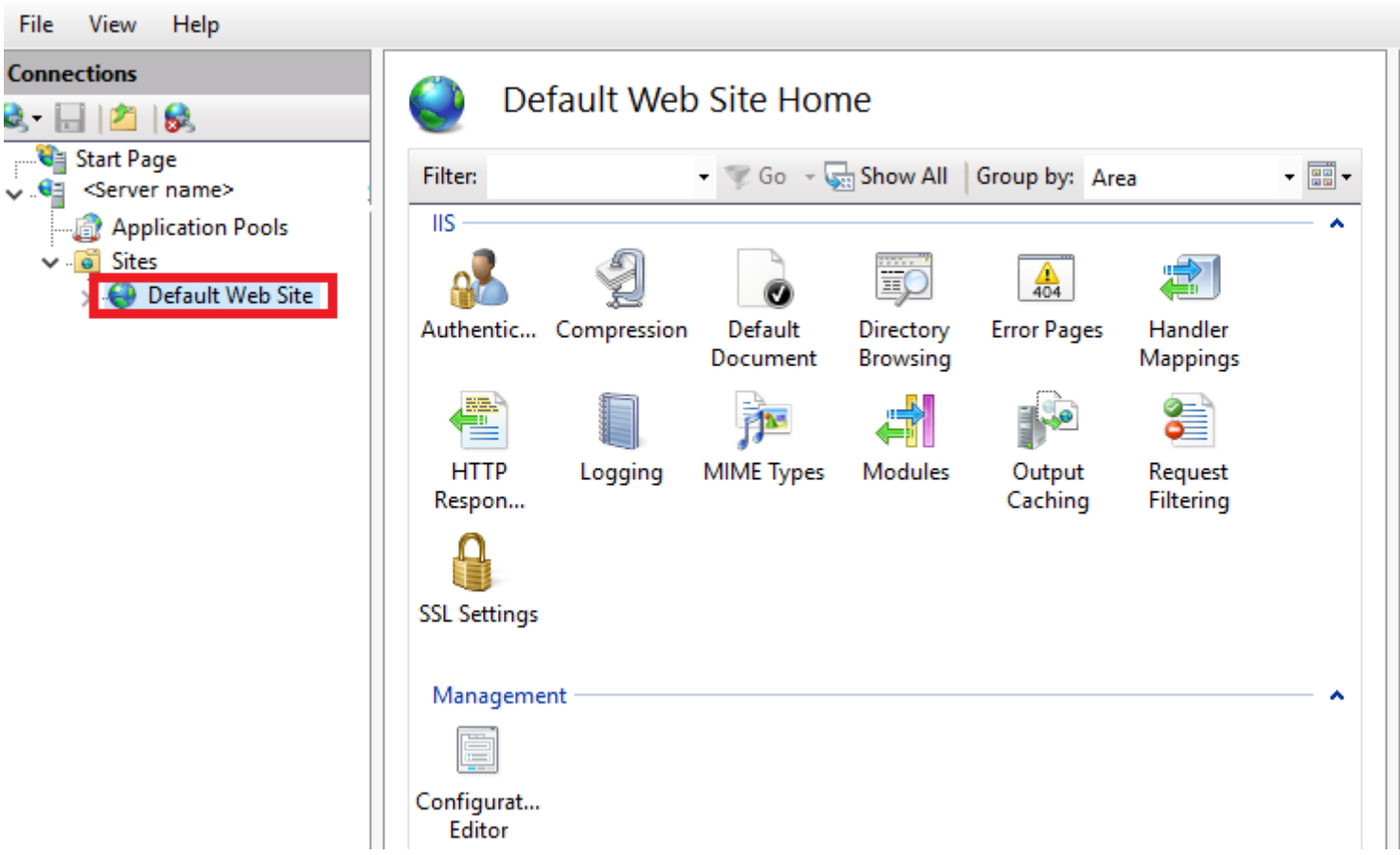
"IIS" is added in the dashboard and can be further configured.



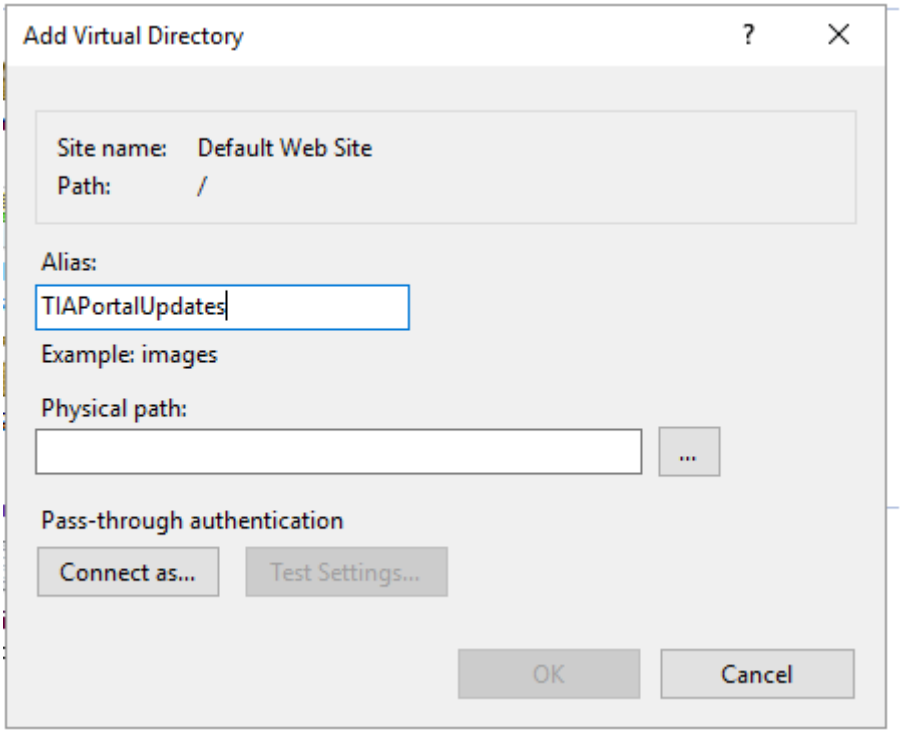
Create website

1. In the navigation area, click "IIS" and right-click in the "Servers" area and select "Internet Information Services (IIS) Manager".
2. In the "Connections" area, click "Sites" and then "Default Web site".

3. Right-click to select the "Add Virtual Directory..." option.



4. In the dialog that opens, enter a display name in the "Alias" field, for example, "TIAPortalUpdates".



5. In the "Physical Path" field, enter the physical path of the folder in which the website is located or click the button to browse (**...**), in order to search for the folder in the file system.

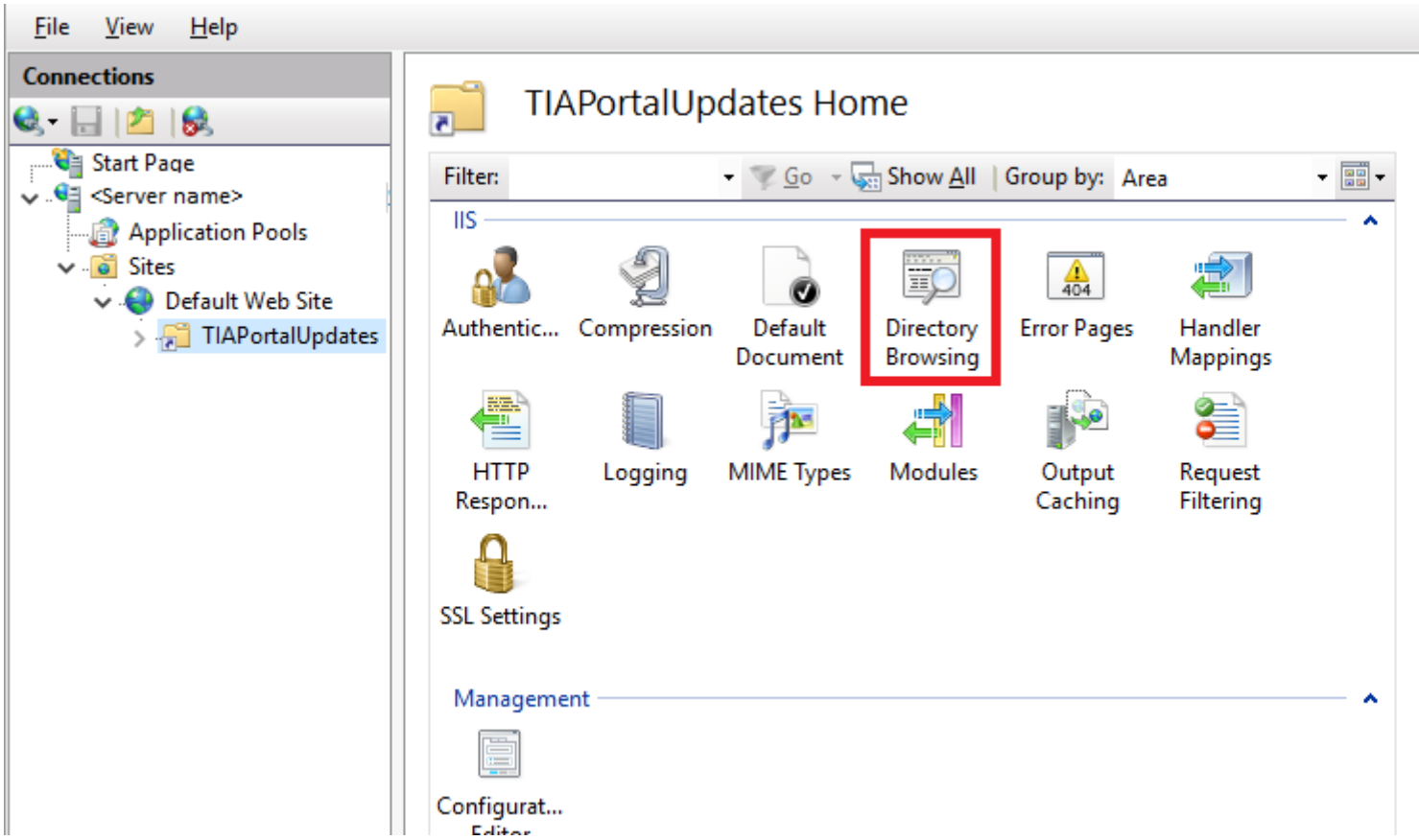
Note

The default website must be created in the directory that contains the 'UpdatesSummaryCatalog.xml' file.

6. Click "Test Settings..." to check whether the settings are correct.
7. Click "OK".

Activating virtual directory

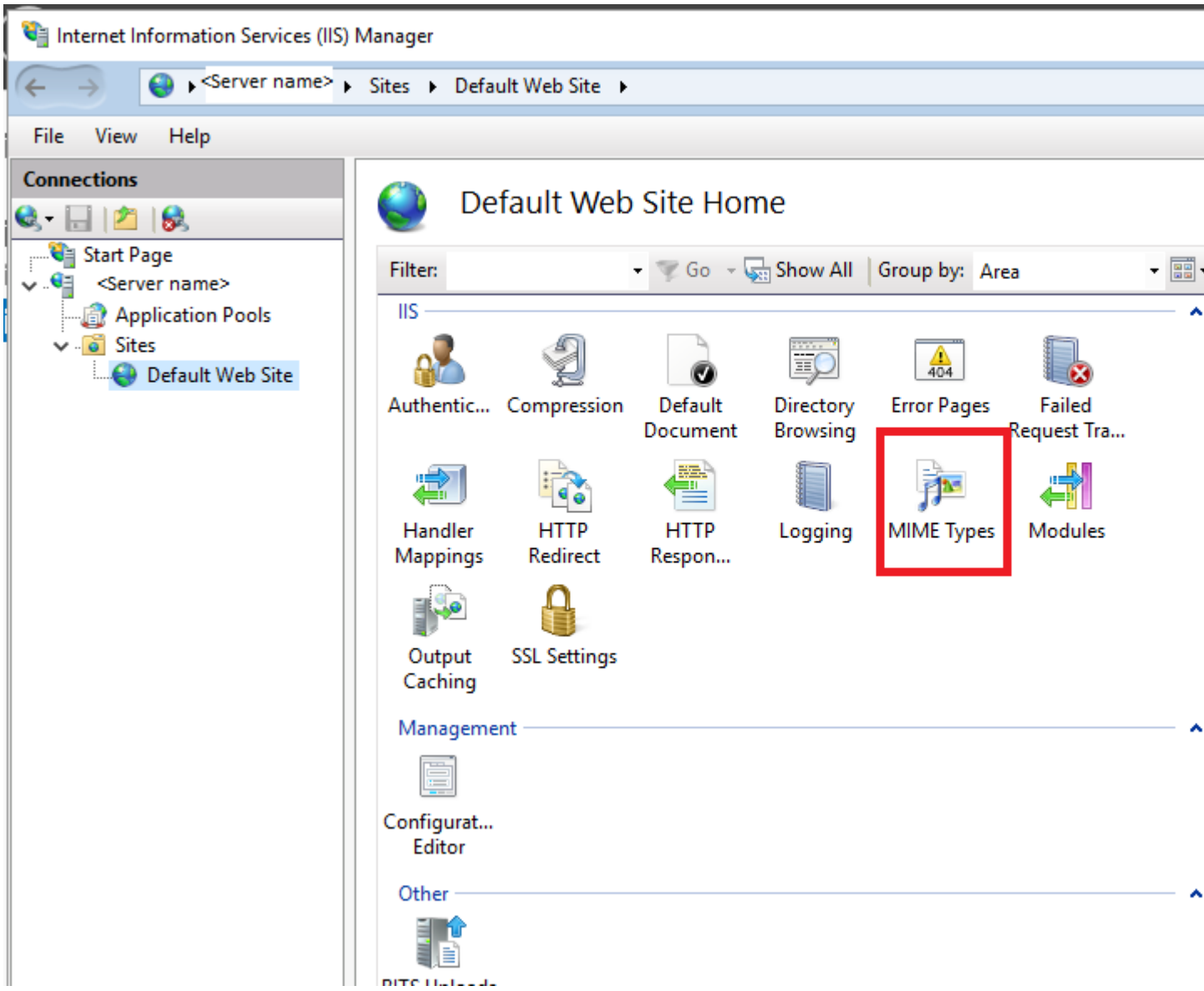
1. In the area "Connections > Sites > Default Web site", select the virtual directory you created.
2. In the "Features" view, double-click the item "Directory browsing".



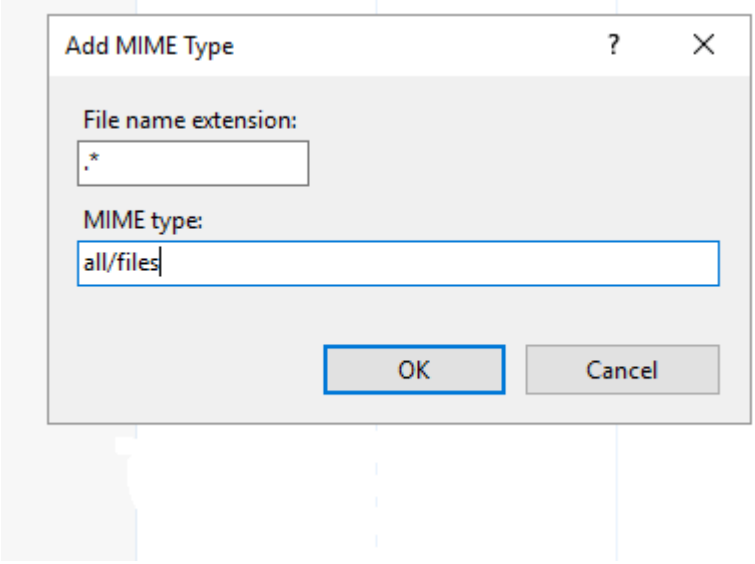
3. Click "Enable" in the "Actions" area.

Adding and configuring MIME type

1. In the "Connections" area, select the website you created.
2. Double-click "MIME type" in the "Features" view.

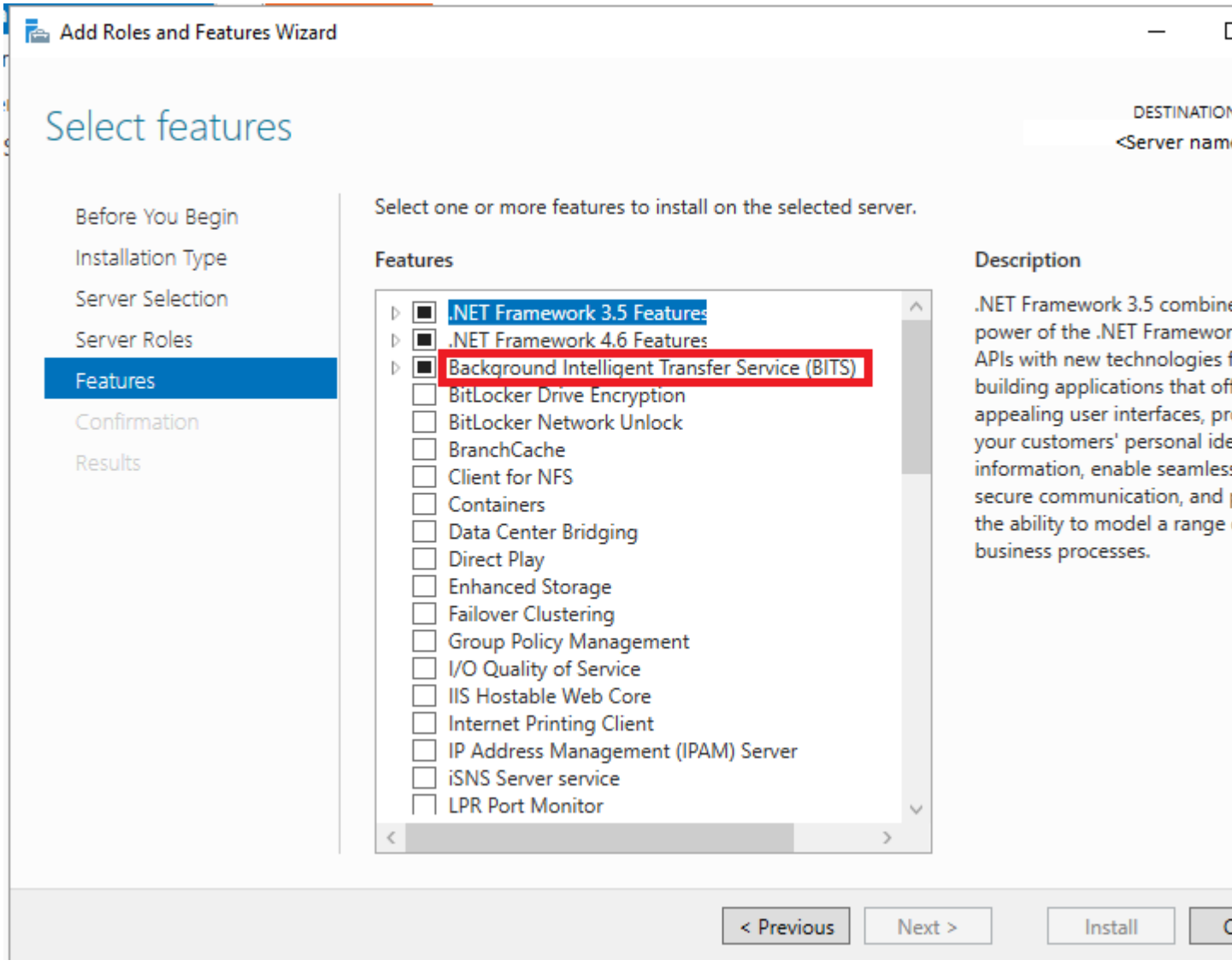


3. Click "Add" in the "Actions" area.
4. Enter .* in the "File name extension" text field in the "Add MIME type" dialog.
5. Enter "all/files" in the "MIME type" text field.
6. Click "OK".

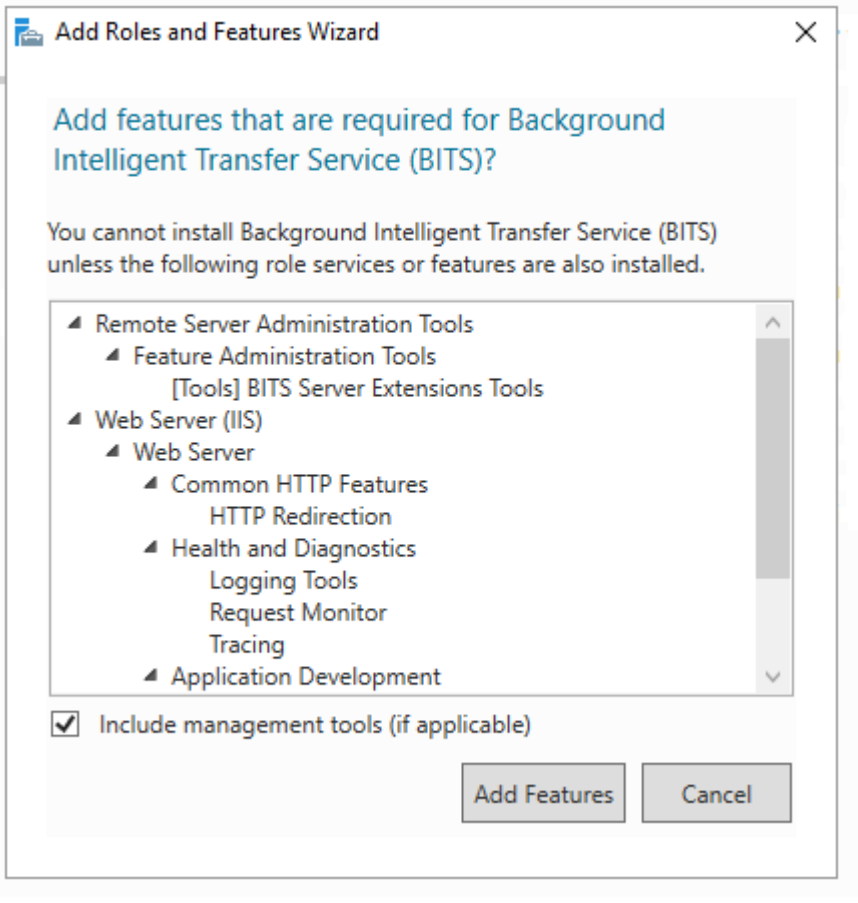

Add MIME Type
File name extension:
.*
MIME type:
all/files
OK
Cancel

Activate BITS-IIS server extension

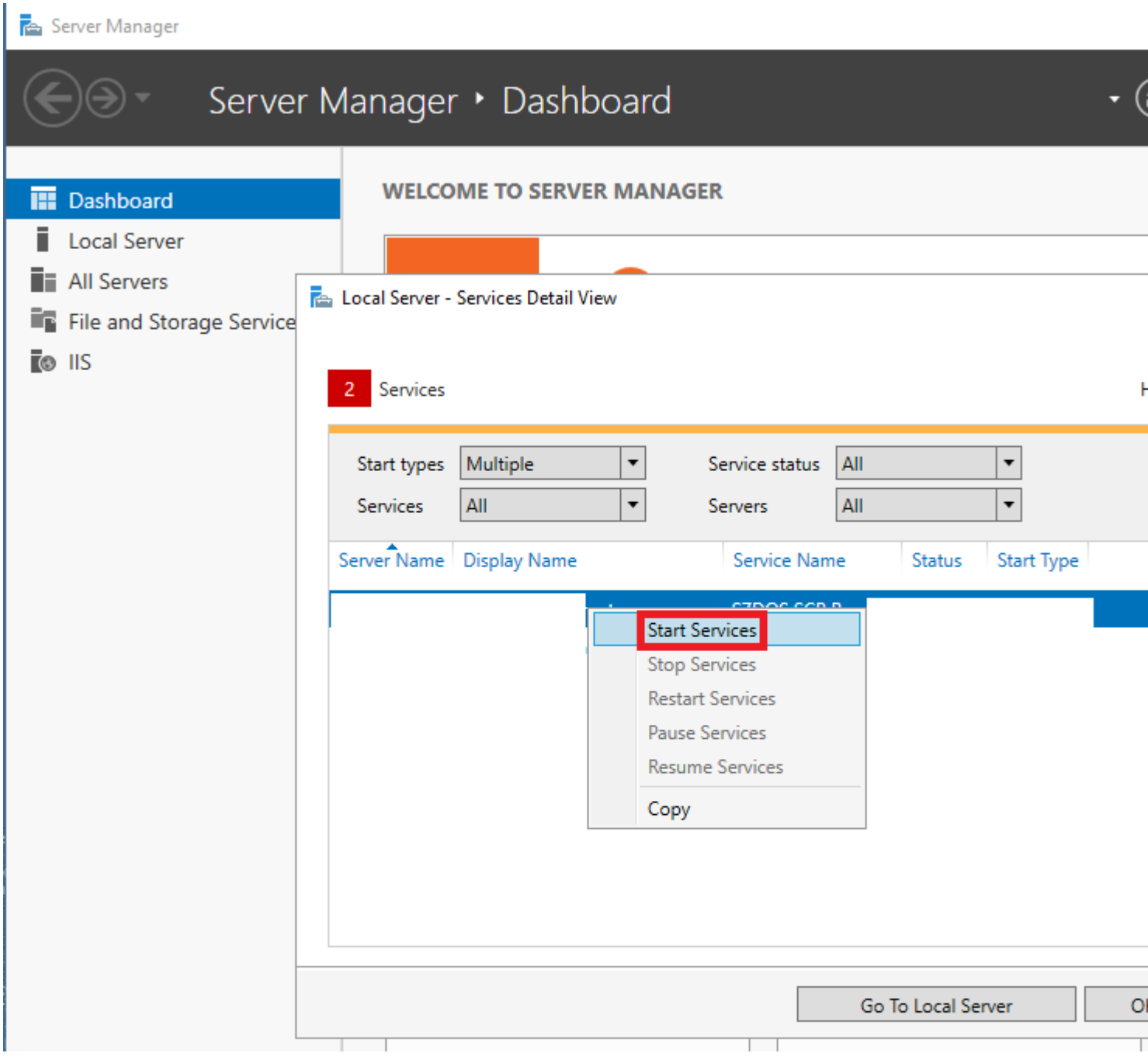
1. Switch to the virtual directory in the IIS Manager.
2. Click "Add roles and features" to verify that BITS is installed. If necessary, enable and install the "Background Intelligent Transfer Service" feature.



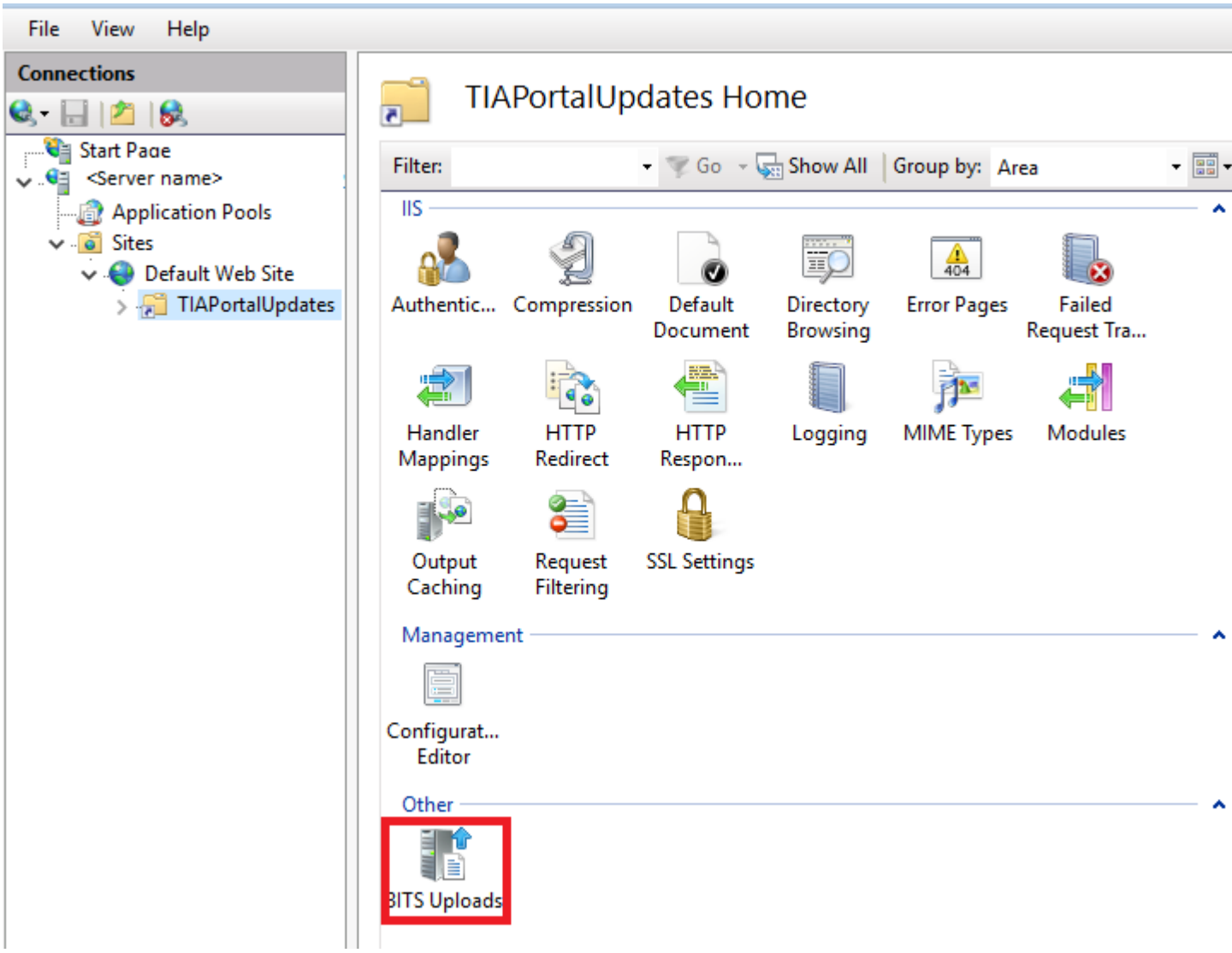
3. In the Dashboard, in the "Local Server" area, click "Services".



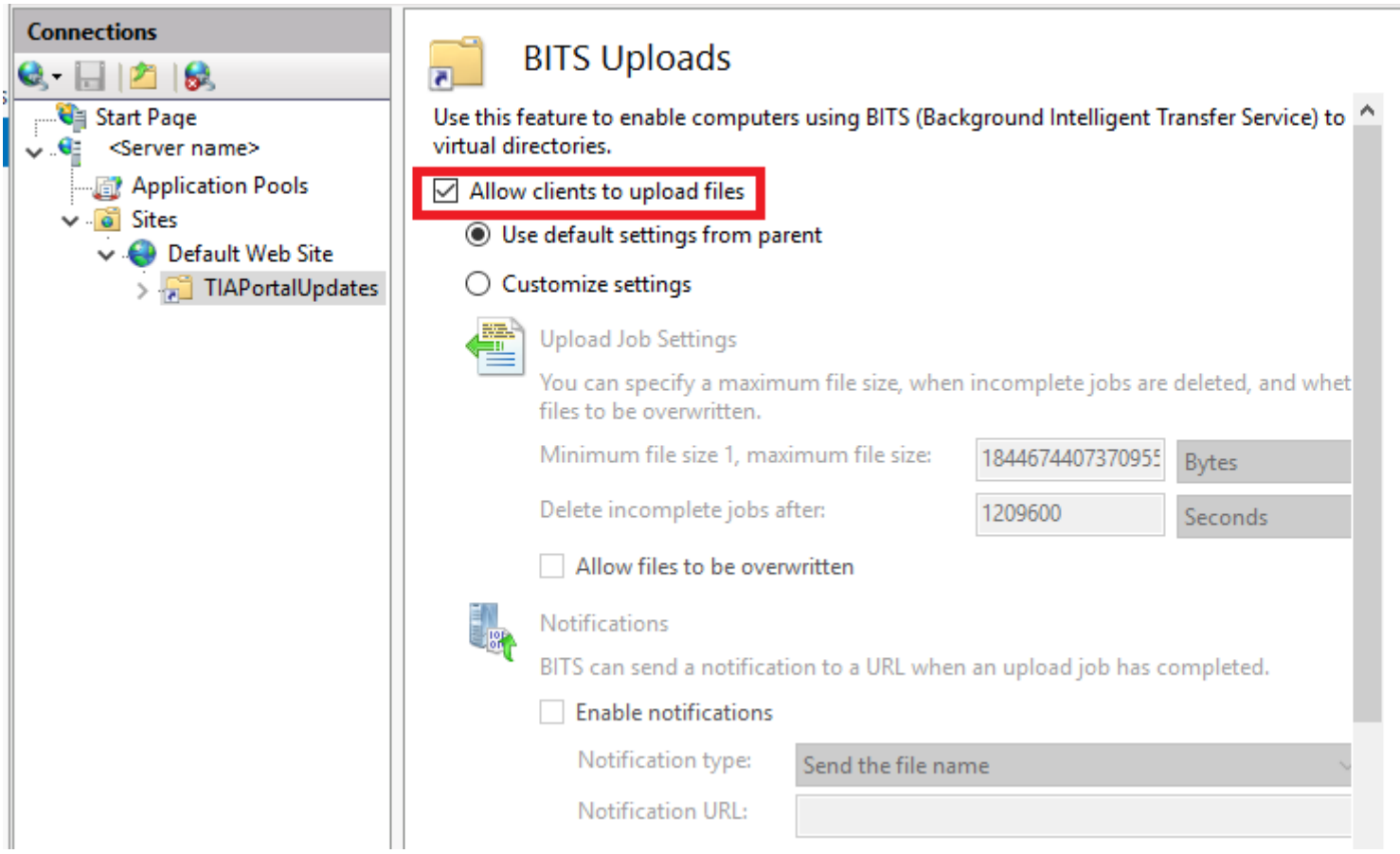
4. Select the local server, right-click and select the menu command "Start Services".


Server Manager
Server Manager ‣ Dashboard
Dashboard
Local Server
All Servers
File and Storage Service
IIS
WELCOME TO SERVER MANAGER
Local Server - Services Detail View
2 Services
Start types Multiple
Service status All
Services All
Servers All
Server Name
Display Name
Service Name
Status
Start Type
Start Services
Stop Services
Restart Services
Pause Services
Resume Services
Copy
Go To Local Server

5. Change to the virtual directory feature view and double-click "BITS Uploads".



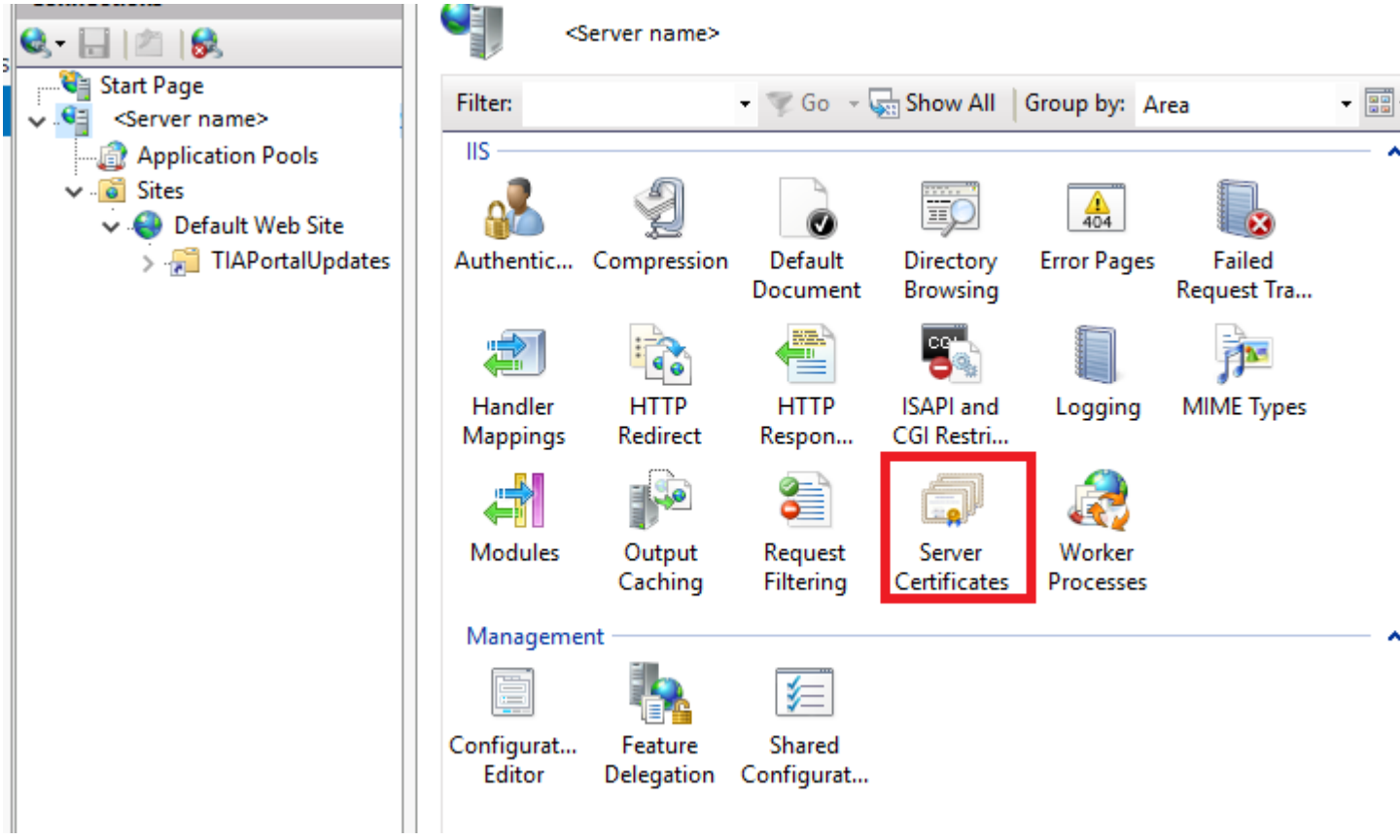
6. Select the "Allow clients to upload files" check box and click "Apply".


Connections
Start Page
<Server name>
Application Pools
Sites
Default Web Site
TIAPortalUpdates
BITS Uploads
Use this feature to enable computers using BITS (Background Intelligent Transfer Service) to virtual directories.
Allow clients to upload files
Use default settings from parent
Customize settings
Upload Job Settings
You can specify a maximum file size, when incomplete jobs are deleted, and whet files to be overwritten.
Minimum file size 1, maximum file size:
1844674407370955
Bytes
Delete incomplete jobs after:
1209600
Seconds
Allow files to be overwritten
Notifications
BITS can send a notification to a URL when an upload job has completed.
Enable notifications
Notification type:
Send the file name
Notification URL:

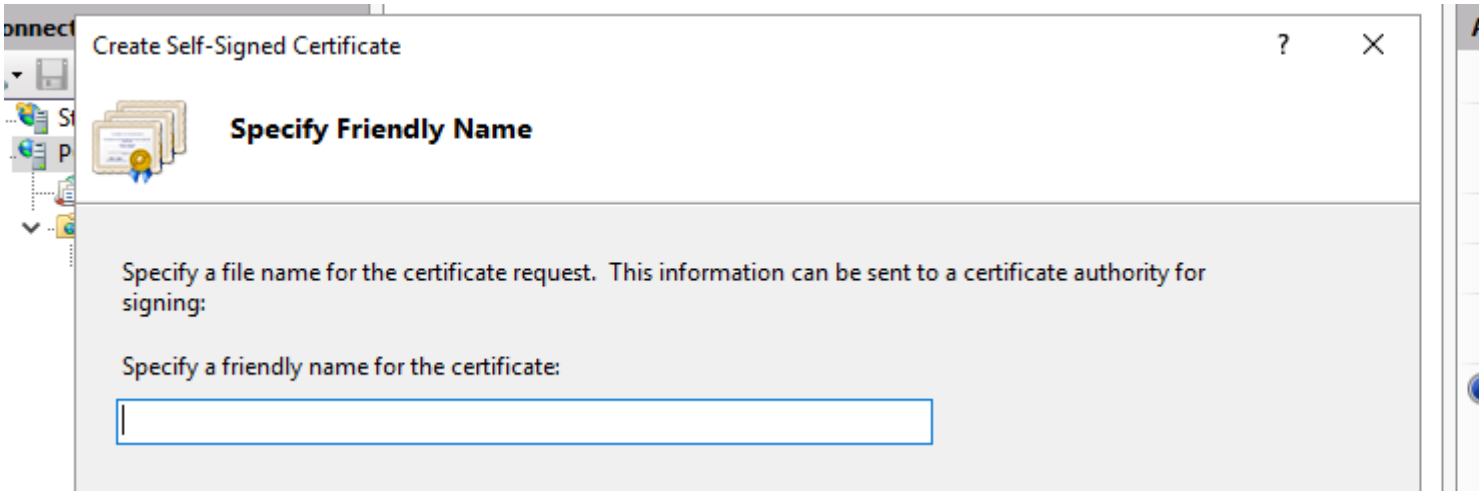
Create a self-signed server certificate

It is strongly recommended that you use a signed certificate created by your company's IT department; in this way the company's own server receives a qualified (secure) certificate. You can also purchase such a certificate from trusted third-party companies/certification authorities. If this is not possible for you, you can create a self-signed certificate. Note that such a certificate is not secure! To do this, follow these steps:

1. In the "Connections" area, navigate to the level which you want to manage.
2. Double-click "Server Certificates" in the "Features" view.



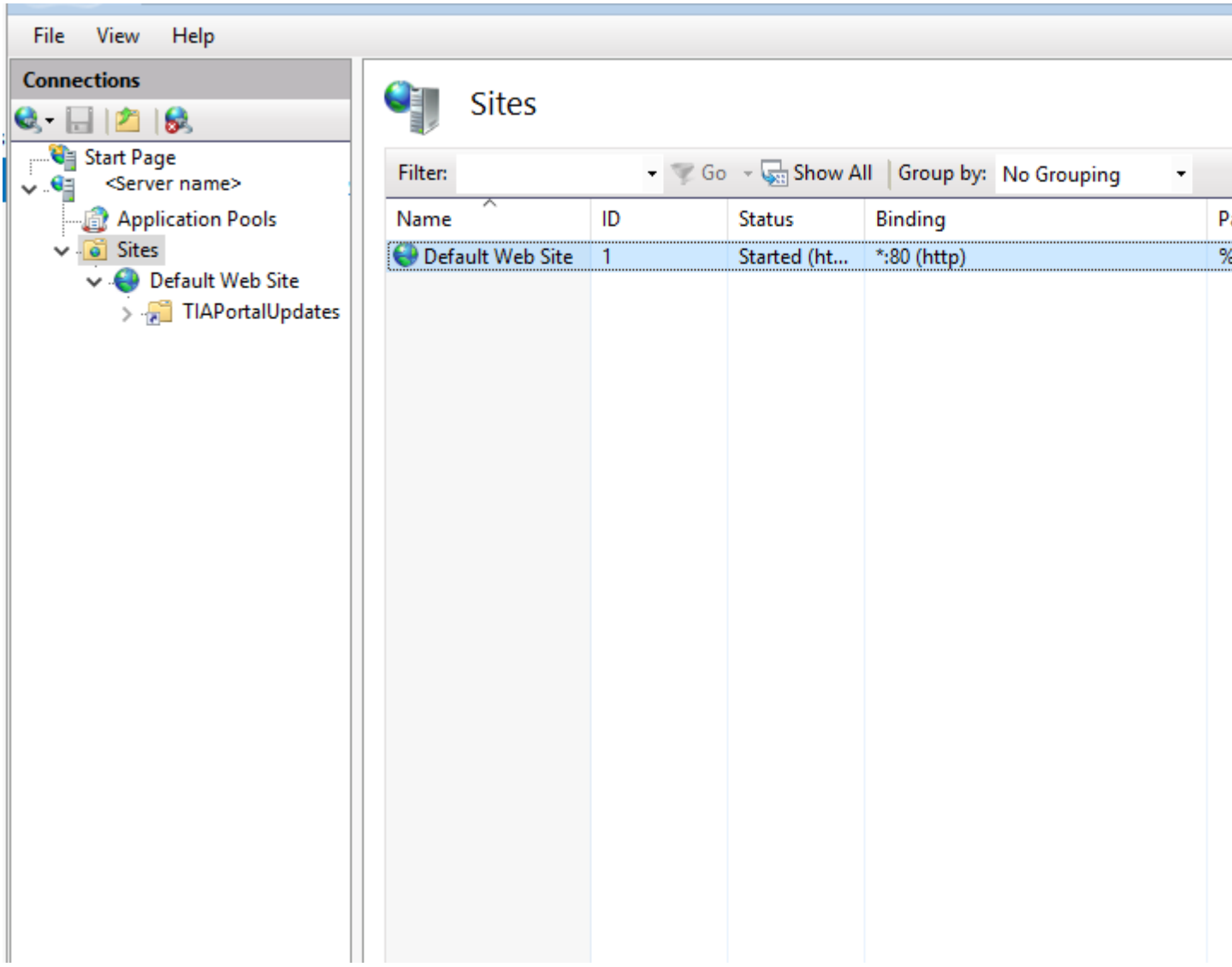
3. Click "Create Self-Signed Certificate" in the "Actions" area.



4. In the "Create Self-Signed certificate" dialog, enter a display name for the certificate in the "Specify a friendly name for the certificate" field and click "OK".

Create SSL binding

1. Expand the "Sites" entry in the "Connections" area, and then click the site to which you want to add a binding.
2. Click "Bindings" in the "Actions" area.



3. In the "Site Bindings" dialog, click "Add".
4. In the "Add Site Binding" dialog, select the certificate you created under "Type" "https" and under "SSL certificate" and then click "OK".

Verify SSL binding

1. In the "Actions" area, click the previously created binding under "Browse website".
   An error page is opened in Internet Explorer, because the self-signed certificate was created by your computer.
2. Click "Continue to this website (not recommended)".
   This message is no longer displayed when you add the certificate to the "Trusted Root Certification Authorities" certificate memory.

Configure SSL settings

1. Double-click "SSL settings" in the "Features" view.
2. Use one of the following procedures in the "Client certificates" area of the "SSL Settings" dialog:
   - Select "Ignore" if no client certificates are to be accepted even if a client has a certificate.
   - Select "Accept" if client certificates are to be accepted.
   - Select "Require" if client certificates are to be required. The "Require SSL" option must be activated to allow the "Require client certificates" option to be used.
3. Click "Apply" in the "Actions" area.

1.12.2.3 Distributing updates to different areas

It may be the case that different departments need different updates or support packages. In this situation, we recommend creating multiple production lines which offer different updates and support packages. The procedure for this is described in the help of the TIA Updater Corporate Configuration Tool.

Note

The main advantage of different production lines is that less storage space is required. When an already downloaded update, support package, or language pack is added to a new production line, a copy of it is not created, instead, the various production lines reference that update, support package, or language pack.

Procedure

You can also operate multiple servers that provide different updates, support packages, and language packs. To do this, follow these steps:

1. Set up different company-owned servers as described above.

or

1. Set up different company-owned servers as described above.
2. Set up multiple websites.
   Make sure that you assign unique names for these websites and the physical paths so that there will be no confusion.
3. Set the features described previously for the Web server or the websites.

You can now store the required updates and support packages for the different departments in the defined directories.

1.12.2.4 Providing updates on a corporate server

Introduction

In the TIA Updater Corporate Configuration Tool, you can set a corporate server on which the available updates/support packages/language packs are stored and can be made available to the users.

Requirement

You must have administrator rights.

A corporate server can only be set up on a Microsoft server operating system, because it requires the operation of the BITS IIS service and a running IIS sever.

Adding updates from the TIA Automation Software Update Server

To add updates from the TIA Automation Software Update Server, follow these steps:

1. Open the TIA Updater Corporate Configuration Tool.
   The available updates are displayed.
2. Click the "Add update" button and select the "Add TIA Automation Software Update Server" check box in the dialog that appears.
3. In the dialog "Add update from TIA Automation Software Update Server", select the required updates (software and support packages) and click "Add".
   If updates are already located on the corporate server, these are grayed out. You cannot load them again. During the download process, the status and the remaining time is displayed in the dialog.

Canceling the download process

To cancel the process, follow these steps:

1. Click "Cancel download".
2. Click "Yes" to confirm the dialog that appears.
   The download of the update is cancelled and it is deleted from the list.

Deleting updates from the corporate server

To delete updates, follow these steps:

Select the required updates and click "Remove".

- If an update only belongs to **one** project/**one** production line, confirm the dialog with "Yes". The selected update is removed from the list and deleted from the file system.
- If an update belongs to **multiple** projects/production lines, the following applies:
  - The remove update dialog does not appear.
  - The selected update is only removed from the current project list and not deleted from the file system.

  To delete updates from the file system, all instances of the updates must be deleted. To do this, follow these steps:
  - Open the existing project/global production line from the "Project" menu.
  - Select the updates you want and click "Remove".
  - Confirm the dialog that appeared when deleting the last instance with "Yes".

Server options

In the dialog "TIA Automation Software Update Server", click "Settings" to determine the following:

1. Under "Server path", specify the folder in which the downloads are to be saved. You can select either a local drive or a network drive.

   Note

   When you work directly on the server, the target directory is the same as the provision directory on the server. All changes are applied directly. This procedure is not recommended, as conflicts can occur with active downloads.

   We recommend that you work on another computer; the target directory can be any directory. The content of this directory must then still be copied to the provision directory. Here, also, make sure that no conflicts occur with active downloads.

2. Select the check box "Always run server in background (icon is shown in the taskbar)", if the server is to always run in the background.
3. Confirm your entries with "OK".

See also

Properties and advantage of a corporate server (Page 94)

Configuring a corporate server for updates (Page 95)

1.13 Installing support packages automatically

1.13.1 Installing support packages automatically

Introduction

As of V15.1, you can use the Support Package Installer to install or upgrade all support packages automatically via the command line; these include HSPs (hardware support packages) that are compatible with the installed version of the TIA Portal (e.g. isp.15_1).

Requirement

- Hardware and software of the programming device or PC meet the system requirements.
- You have administrator privileges on your computer.
- All running programs have been closed.

Procedure

To start the installation with the desired options directly via the command line, proceed as follows:

1. Open the Windows command prompt with "Start > Run > cmd".
2. Switch to the directory that contains the "Siemens.Automation.SupportPackageInstaller.exe" file. This is the installation directory of your installed TIA Portal.
3. In the command prompt, enter the following command:
   %Installation directory% > Siemens.Automation.SupportPackageInstaller.exe <Support Package(HSP)-storage directory> [-l <Name of log file>] [-warnaserror]

<Support Package-storage_directory>
Mandatory directory containing the support package files (*.isp). This can be either a local folder or a Remote Share.

-l <Storage location of log file>
This parameter is optional and specifies the path to the log file. If it is not set, the log file is created in the binary directory. However, you need to have write permissions in the storage folder.

-warnaserror
This parameter is optional and indicates that every message that would usually have been reported as a warning will be reported as an error instead. If a HSP does not meet the necessary conditions, for example, a corresponding return value/abort value is output to the command line and the affected HSP is skipped.

-?
This parameter is optional and shows the Help.

Note

- If the storage directory or the name of the log file contains spaces, you need to place the name in quotation marks.
- Only support packages located directly in the storage directory are considered; subdirectories are ignored. This means that the *.isp files must be located directly in the storage directory.

See also

Return values from the installation process (Page 118)

Log file (Page 119)

1.13.2 Return values from the installation process

Return values from the installation process

The following table shows the return values and their descriptions:

Code	Description
0	The installation was successful.
1	The installation was successful and a restart is required.
-1	The installation failed.
-2	The installation failed: The TIA Portal is still running.
-3	The installation failed: Another installation is already running.
-4	The installation failed: You do not have sufficient rights. The installation requires administrator rights.
-6	The installation failed: Insufficient memory space.
-7	The installation failed: A restart is required before the installation can start.

See also

Installing support packages automatically (Page 117)

Log file (Page 119)

1.13.3 Log file

Contents of the log file

The log file contains detailed information about every installation. If the log file already exists, it is supplemented. Headers and footers are created for each log file between which you can find the desired information.

An entry is generated for the following events:

- Support packages that are listed in the storage directory and are available during the installation but could not be installed due to missing products. The entry in the log file indicates which products and which versions of these products are missing.
- Support packages that could not be installed successfully.
- Support packages that were installed successfully.

The file name of the HSP including the file extension is logged for each entry.

See also

Installing support packages automatically (Page 117)

Return values from the installation process (Page 118)

Read me

2

2.1 Security information (Unified)

Security information

Siemens provides products and solutions with industrial security functions that support secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions only form one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary, and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

You can find more information on the protective measures possible in the area of industrial security at:

https://www.siemens.com/industrialsecurity

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends applying product updates as soon as they are available and always using only the latest product versions. Use of product versions that are no longer supported, and failure to apply latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under

https://www.siemens.com/cert

Network settings

The following table shows the network ports that are used by WinCC Unified for internal and external communication. These ports must not be used for any other purpose.

The setup configures the firewall to ensure smooth operation.

WinCC Unified		
Name	**Port number**	**Transport protocol**
ILScs Manager	20008	TCP
UMC	20009	TCP
ILPmon Manager	4999	TCP
ILEvent Manager	1235	TCP
ILDist Manager	4777	TCP

WinCC Unified		
ILDataManager	1234 5001	TCP
Node Processes	3103 443 8888	TCP
Graphics Runtime	1339 1345	TCP
License server	1366	TCP
Screen debugger	9222	TCP
Job debugger	9224	TCP
WCCIL Proxy Manager	5678	TCP
Network Discovery (IS)	137	TCP
UMC AttachAgent	4002	TCP
OPC UA Discovery	4840	TCP
RFID	5003	TCP
GraphQL	4000	TCP

Note

Block HTTP port

For security reasons, it is recommended that Port 80 be disabled on the IIS server.

1. Start "Internet Information Services (IIS) Manager" on the Unified PC.
2. Right-click "Default Web Site".
3. Select "Remove" or "Manage Website > Stop".

2.2 Breaking changes

PI options in ODK

ODK does not support PI options in V18.

Scrolling behavior for pop-ups

If you move the visible screen section of a process screen in Runtime and a pop-up window is visible there, for example a faceplate container, Runtime behaves as follows:

WinCC Unified V17	The pop-up window is moved together with the screen section.
As of WinCC Unified V18	The pop-up window is not moved with it.
Simulation of an HMI device with installed device version V17 in a WinCC project V18	The pop-up window is moved together with the screen section.

Operation of objects behind a screen window

If a screen window has an empty area and the screen or faceplate displayed in the screen window has the "transparent" setting as the fill pattern for the background, Runtime behaves as follows:

WinCC Unified V17	Objects that are in the background of the screen window in the empty area of the screen window are visible. The objects cannot be operated by clicking.
As of WinCC Unified V18	The objects can be operated by clicking. Application example: You use a round menu with a button next to it. To prevent the rectangular border surrounding the menu from obscuring the button, make the menu available in a screen window.

Zoom behavior of screen windows on touch devices

Runtime now supports panning and zooming with a two-finger gesture.

WinCC Unified V17	No panning and zooming with two-finger gestures.
As of WinCC Unified V18	Panning and zooming with two-finger gestures is supported. Default: Activated If you want, you can disable this feature by clearing "Format > Allow zoom" property for the screen window.

Exclusion of "Uncertain - initial value" quality code as trigger

WinCC Unified V17	If a tag is used as a trigger for a script and the tag is used in a screen, e.g. by an I/O field, the script is executed twice in quick succession when the screen is loaded: • When reporting the initial value • When reporting the process value
As of WinCC Unified V18	Default setting: QualityCode 0x4C (Quality status: Uncertain, Substatus: Initial value) is ignored as a tag trigger. If a tag is used as a trigger for a script and the variable is used in a screen, e.g. by an I/O field, the script is only executed when the process value is reported as the screen is loaded. To enable the quality code as a trigger, proceed as described below.

To enable quality code 0xC4 as a trigger in V18, do the following:

1. Open a file browser and navigate to the following folder:
   `C:\Program Files\Siemens\Automation\WinCCUnified\bin\_config`
2. Open the following configuration file:
   - For scripts in screens: `IOWA_GfxComponentConfigurationSrv.xml`
   - For scheduled tasks: `IOWA_SchedComponentConfiguration.xml`
3. Delete the line for <Attribute key>:

```
<ComponentConfiguration componentKey="GfxTriggerManager"
name="Trigger Manager">
   <AttributeList>
      <Attribute key="IgnoredTagQualities" value="0x4c"/> <!–
0x4c: "Initial Value" quality >
   </AttributeList>
</ComponentConfiguration>
```

Simulation of Runtime

To use the simulation, you must install WinCC Unified Runtime. If no license is found for Runtime, start the simulation in demo mode.

2.3 Notes on use

Contents

Information that could not be included in the online help and important information about product features.

Previously installed versions of WinCC Unified

The installation of Unified V18 is possible on devices for which the following applies:

- No Unified installed yet, or
- WinCC Unified V17 installed
  The V17 installation must not have been upgraded from V16.

Special characters in the installation path

Special characters in the installation path may lead to objects not being visible when inserted into a screen. To see the object, you have to close the screen and open it again.

Copying between WinCC Unified and other devices

Copying data via the clipboard or via the library, for example is only supported between Unified devices. Copying data between Unified devices and other devices is not supported.

Secure communication

Secure communication via the TLS protocol can only be established via integrated connections.

If you use Inter-Project Engineering (IPE) in your project, i.e. an HMI device is connected to a device proxy PLC, and an error message about inconsistent certificates appears during compiling, you need to correct the certificates of the associated PLC in the original project. Then you have to compile, export and re-import the PLC.

Changing HMI device names or log paths

If you make the following changes in the engineering system for a project that is already loaded in Runtime, the existing log data is lost when you load the device again:

- You rename the HMI device in the project navigation
- You rename a log folder or change the log path in the Runtime settings

The data from the old log is not available in the new log after reloading.

Access to subnets via TCP/IP Auto

Use the TCP/IP Auto setting on your engineering device (programming device) when you connect the device to a local subnet. (Control Panel > Set PG/PC interface)

The engineering device then allows additional IP addresses from the subnet to be added to its network adapter.

Note

This setting cannot be used if you use a router.

No setting is necessary on the devices belonging to the external network.

For stationary operation, use the TCPI/IP setting.

Fault in the connection between the server and the client

If there is a fault in the connection between the server and client, check the settings of the PG/PC interface. TCP/IP Auto should not be used for the setting "Interface parameter assignment used". Use fixed IP addresses instead.

Changes at a PLC user data type

After a change to a PLC user data type, the PLC must be compiled to make this change known to all Unified devices that use this user data type. If compilation of the PLC does not take place, error messages can occur during the compilation of the HMI devices.

Inter Project Engineering (IPE) in faceplates

Inter Project Engineering is not supported by faceplates. User data types of a device proxy PLC cannot be interconnected with a faceplate.

Certificate configuration after upgrading a Runtime Collaboration device

The certificate configuration created in V16 cannot be reused in V17. For both upgraded devices and devices newly introduced in V17, the certification authority (CA) including certificate configuration must be created again using the WinCC Unified Certificate Manager V17 tool.

1. Create a new certification authority.
2. Add the devices from V16 to the new certification authority.
3. Add new V17 devices.
4. Create the certificates of the devices.
5. Distribute and install the certificate configuration to the devices.

Starting Runtime

Runtime cannot be started via WinCC Runtime Start if the "Load preview" dialog is displayed in the engineering system.

Status "Partly running"

If it is not possible to start the simulation or the Unified Runtime, open the Runtime Manager. If the status of the project is displayed as "partly running", check

- whether the user currently logged on in Runtime has sufficient rights. Is the user entered in the following Windows user groups:
  PLCSimUsers
  RTIL Tracing Users
  Siemens TIA Engineer
  SIMATIC HMI
  SIMATIC HMI VIEWER
- whether the computer name is not longer than 15 characters.
- whether "OPC UA" is activated in the Runtime settings and a certificate exists.
- whether "Runtime Collaboration" is enabled in the Runtime settings of a Unified Panel and a certificate is available.

Simulation with S7-PLCSIM

Always start the simulation in WinCC and then S7-PLCSIM.

WinCC Unified Tag Simulator

The WinCC Unified Tag Simulator is not part of WinCC Unified V18.

Process control

At this time, no archive values can be deleted with the "Delete archive value" button.

Dynamic SVG types

If you manually assign a type version to a dynamic SVG type or rename the type, a delta download is no longer possible.

Encryption with TLS

Always use the most current version of TLS. Disable the older version.

The use of older versions (TLS 1.0 and 1.1) is at your own risk.

Assigning text lists and graphics lists for reports

It is not possible to read in values from texts lists and graphics lists, neither in the add-in nor when generating the report.

Page numbering in audit

Because of a problem with the page numbering, the system functions "ReadElectronicRecord" and "ExportElectronicRecordAsCSV" only deliver the first 1000 electronic data records on the provided filter.

GraphQL

The GraphQL interface only supports Unified PC in V18.

2.4 Screens and screen objects

Contents

Information that could not be included in the online help and important information about product features.

Screen objects for Unified Comfort Panel

These following objects are supported in WinCC only for Unified PC.

- Plant overview
- Process control

If you have configured the plant overview or the process control on a Unified Comfort Panel, you must delete these objects before the compilation.

Adaptations to screens and screen objects

The settings that you define under "Settings > Visualization" in the "Resize screen" area have no effect on the HMI devices:

- SIMATIC Unified PC
- SIMATIC Unified Comfort Panel

Trend control

In a trend control with several configured trend areas, the buttons "First data record" and "Last data record" always act only on the last trend area that was configured. In other trend areas, you can use the "Select time range" button to navigate to the first or last data record.
If you have to navigate to the first or last data record very frequently in all configured trend areas, it is advisable to configure the trend area in several trend controls.

Process control

At this time, no archive values can be deleted with the "Delete archive value" button.

System diagnostics

The "System diagnostics display" control supports the S7-1500 as of firmware version 2.0. The prerequisite is that the "Central alarm management in the PLC" setting is activated at the PLC.

If you configure HMI connections to more than 50 PLCs for the control, this can lead to an overload in Runtime. As a result, HMI connections can no longer be established.

Process diagnostics

- You require the WinCC Unified ProDiag license to use the "Process diagnostics" control.
- The control supports S7 GRAPH instance data blocks as of version 6.0.
- The button for activating/deactivating the criteria analysis is reserved for future versions.
- With the following buttons you navigate to the previous or next sequence of the displayed S7 GRAPH block, not as described in the user help to the previous or next network:

Clock

The "Clock" object does not have a property that sets the design of the clock as a digital display or as an analog display. When using the light or dark style, the digital display is used if the clock is configured smaller than 100 x 100 pixels.

SVG graphics

SVG graphics and dynamic SVG graphics support the SVG 1.2 Tiny standard. The use of SVG graphics that include the elements or commands according to the SVG 2.0 standard can lead to unexpected behavior.

The file names of SVG graphics must not include German umlauts or Chinese characters.

No embedding of session-local tags in dynamic information

Alarms and screens can contain texts in with dynamic information is embedded. This dynamic information is read from alarm parameters, text lists or tags.
Examples of texts with dynamic information: Alarm texts, text list entries, the "Text" property of screen elements such as buttons

Session local variables are not supported as a source of such dynamic information.

Faceplates: Passing interface properties of the "Resource list" data type to pop-ups

When you call pop-up windows outside a faceplate type, you pass the interface properties by script. For interface properties of the "Resource List" data type, text and graphic lists can be passed.

To specify graphic lists, use the prefix "GraphicListCollection", for example, "GraphicListCollection.Graphic_List_1".

To specify text lists, use "@Default" as a prefix, e.g. "@Default.Text_list_1".

Faceplates: Passing interface properties of the "Multilingual text" data type to pop-ups

When you call pop-up windows outside a faceplate type, you pass the interface properties by script. Interface properties of the "Multilingual text" data type cannot be passed.

In this case, use the interface property of the "configuration string" data type. The configuration string contains the different values of the text in the respective language separated by a semicolon. Use the script to read the active Runtime language and then use the "Case" command to make a case distinction depending on the active Runtime language.

Faceplates: "Cleared" event

The "Cleared" event occurs when the faceplate is closed. The relevant faceplate container is dissolved.

Therefore, no interface tags or interface properties can be passed on the "Cleared" event.

2.5 Alarms and alarm view

Contents

Information that could not be included in the online help and important information about product features.

Alarms and alarm view

Controller alarms

If the connection between HMI device and controller is interrupted, only the last occurring controller message can be sent and displayed after restarting the runtime software. When the connection is re-established, all controller messages are transmitted correctly again.

Alarm block "Duration"

Alarm block "Duration" is not supported in V17.

2.6 "Smoothing" property for logging tags

Contents

Information that could not be included in the online help and important information about product features.

Smoothing (Unified Comfort Panel)

The following applies to logging tags and PLC tags: If the value "No smoothing" is set in the Logging tag properties under "Smoothing > Mode", the values are nevertheless smoothed.

Example:

A logging tag changes its value as follows: "100" > "101" > "101".

Even if "No smoothing" is set in the properties of the tag, the values [100, 101] are stored in the log.

2.7 System functions and scripts

Contents

Information that could not be included in the online help and important information about product features.

JavaScript arguments of specific object events

The following applies to the rectangle, gauge, button and circle objects: If you configure a JavaScript to an event, only the "Item" argument is passed correctly to the script. You can process the event in the script, but not the other arguments.

"StartProgram" in the Scheduler - Unified Comfort Panel

Calling up a program via the "StartProgram" system function in the Scheduler is not supported by Unified Comfort Panel. Instead, use a "Tag" trigger in the Scheduler to initiate the system function.

"Press key" and "Release key" events - Unified Comfort Panel

The events "Press key" and "Release key" are not supported on a Unified Comfort Panel.

Methods of the "ProDiag" object

With the "OpenTIAPortal" methods of the "ProDiag" object, TIA Portal can be opened from the Runtime. Prerequisite for this is that the "Allow start of external processes via Unified Runtime" setting is activated on the HMI device in SIMATIC Runtime Manager.

The following system functions are reserved for future versions:

- "ResetToConfiguration"
- "OpenPlcCodeViewByFCCall"
- "OpenProDiagDetailsByCall"

With the parameter "screenItemPath" of the " IsJumpableAlarm" method, you transfer the path of the screen object which is activated when the alarm that is selected in the alarm display is a ProDiag alarm – not the path of the code viewer as described in the user help. Example: Transfer the path of a button for which a jump into the PLC code display was configured. If the selected alarm is a ProDiag alarm, operators jump to the PLC code display in Runtime by clicking the button.

The system function "OpenViewerGraphByBlock" expects the name of the data block of the GRAPH function block as second parameter.

ExportParameterSets

The system function "ExportParameterSets" expects as third parameter the path plus file name of the file to which the ParameterSets are exported.

Uniform display of custom web controls

To make the display of a Custom Web Control independent of the browser you are using, you must use a stylesheet when programming the Custom Web Control.

2.8 Parameter sets and parameter set display

Contents

Information that could not be included in the online help and important information about product features.

Using parameter set display from V16 project

If you have configured a parameter set control in WinCC V16 without updates, open this project with WinCC V17 and download it to the HMI device, then the numbers of the parameter sets are not displayed on the HMI device. Proceed as follows to continue using this parameter set control:

1. Create a new project in a new instance of WinCC V16 with installed update.
2. Insert a Unified Comfort Panel or a SIMATIC Unified PC into the project.
3. Copy the devices from the original project into the new project.

Configuring a storage medium for parameter sets

Select the same storage medium that you have configured for tag persistence or alarm logging under "Properties > Information" for parameter set types. If you specify different storage media, the "EjectStorageMedium" system function is not available.

2.9 WinCC Unified PC

2.9.1 Notes on the operation of Unified PC

Contents

Information that could not be included in the online help and important information about product features.

Generic logon error due to browser language settings

If a language that is not supported by Unified Runtime is set as the browser language, the "Generic error" occurs during logon.

Follow these steps:

1. Open the browser language settings.
2. Select one of the languages supported by Unified Runtime.
3. Log on in Runtime.

Restoring log segments

The following requirements apply to restoring log segments with SIMATIC Runtime Manager:

- A project that is in the "Running" status is loaded into runtime.
- At least one backup of a tag or alarm log is available for the project.
- If the project was loaded several times, no logs were reset during complete loading in the "Load preview" dialog.

TraceViewer

If the user logged on in Runtime belongs to the "RTIL Tracing Users" and "SIMATIC HMI" user groups, all trace messages are visible in the TraceViewer, otherwise only the trace messages from SIMATIC Runtime Manager.

2.9.2 Internet browsers for WinCC Unified PC

Ensure you have the latest operating system and browser version if you want to access Runtime Unified with this device.

WinCC Unified displays the runtime elements in HTML5. The browser used also has to support this standard. Since the browsers interpret HTML5 differently, it is possible that objects are displayed differently depending on the browser and the browser version used. For example, browsers sometimes display fonts differently.

Compatibility tests were performed for the following browsers. The focus of the compatibility tests was on the browsers marked with *:

Operating system	Browser
Microsoft Windows	• Google Chrome* • Microsoft Edge • Mozilla Firefox, Mozilla Firefox ESR
Android	• Google Chrome* • Firefox • Edge
iOS, Mac	• Safari* • Google Chrome • Firefox • Edge

Browser recommendation

In view of the performance and support of the Runtime standard elements, Google Chrome has proven to be the preferred browser. Its memory requirements are slightly higher than those of the other browsers.

Note

Operating system and browser version

For Runtime operation via Android or iOS, always use the latest operating system.

Use the latest browser version.

Note

Performance differences in different versions of individual browsers

The browser versions can differ from each other, which can result in different behavior regarding the memory requirements and speed.

Note

Suitability for continuous operation

MS Edge and Mozilla Firefox have proven to be problematic in continuous operation.

Known browser problems

The following restrictions apply to the following browsers:

Internet browser	Limitation
MS Edge	• High memory capacity utilization in continuous operation
Mozilla Firefox	• High memory capacity utilization in continuous operation
Mozilla Firefox ESR	• Support of touch gestures for touch panels as of Firefox ESR V59
Google Chrome	• High memory capacity utilization in uninterrupted duty depending on the version. • On Android: Grid lines with a line width ≤1 are not displayed correctly. This is due to the browser's own line thickness representation. As a solution, it is helpful to use a line width ≥1. • No correct representation of elements that use an SVG graphic as background graphic scaled in the Engineering System.

Restrictions to the various functions can also occur, such as the availability of the before and after buttons in the controls.

Current information on browser problems

You can find up-to-date information on display problems in browsers at the Siemens Online Support under the entry ID 109757952.

2.9.3 Activating and testing ASIA licenses

Overview

The license keys for WinCC Runtime Unified are available on the "License Key USB Hardlock" license storage medium.

The licensed ASIA version is executable in parallel to the European version by switching to Unicode.

The "License Key USB Hardlock" (dongle) checks the following conditions:

- WinCC GUI language
- Runtime language
- Asian characters are used in the WinCC project.
- Operating system settings

Note

It is not allowed to run WinCC in process mode without a valid license.

Delete configuration languages

If you do not have a license for an ASIA version and delete the Asian configuration languages, the WinCC project continues to run in demo mode.

To disable Demo mode, close the WinCC project. When reopened it is recognized that the WinCC project no longer requires licenses for an ASIA version.

Testing the validity of the licenses

If you start a correctly licensed WinCC version without the connected dongle, an error message appears. The same error message appears after a few minutes if you disconnect the dongle from the computer with a correctly licensed WinCC version.

If this error message does not appear, a non-licensed WinCC version is installed. No right of usage for WinCC is available in this case. Uninstall this WinCC version and purchase a legally licensed version of WinCC.

If necessary, contact WinCC Support and provide the serial number of your software version:

- http://www.automation.siemens.com/partner/index.asp

You can find the serial number on the "Certificate of License" (CoL).

Working with the "License Key USB Hardlock"

Please note the following:

- Do not edit data on the "License Key USB Hardlock".
  The actions not allowed include:
  - Rename data
  - Delete data
  - Copy data to the "License Key USB Hardlock"
- Do not format the "License Key USB Hardlock".
- Do not remove the "License Key USB Hardlock" from the PC while WinCC is running.

NOTICE
Do not remove the "License Key USB Hardlock" dongle If you remove the dongle from the computer, an error message is generated and WinCC switches to Demo mode. If you re-connect the dongle to the computer, the error message disappears and Demo mode is disabled. WinCC works once again in licensed mode.

2.10 Notes on the operation of Unified Comfort Panel

Contents

Information that could not be included in the online help and important information about product features.

Printing on a Unified Comfort Panel

With the objects "Trend control", "Function trend control", "Value table", Unified Comfort Panel supports printing via the button in the toolbar. The data to be printed are sent to the specified standard printer in the Control Panel. The last 10 print orders are saved as a graphic on the panel in the directory /home/industrial/ControlScreenshot.

Changes to the settings of the SOC1 interface (X1)

The following changes to the settings of the SOC1 interface (X1) require a manual restart of the Panel:

- Settings related to the mode and speed of the SOC1 ports
- Media redundancy settings
- Deactivating the forwarding of DCP broadcasts
- Deactivating the sending of LLDP

Screen objects for Unified Comfort Panel

The following screen objects are not supported by Unified Comfort Panel:

- Process control

Hotkeys for buttons

The "Hotkeys" functionality is not supported on the Unified Comfort Panel for the "Object" button and the buttons in other objects.

Objects with buttons

Some objects such as the trend control or the parameter set control have buttons for which you can configure access rights. To operate access-protected buttons, a user with the required authorization must log on via the Control Panel or a configured operating object.

"Press key" and "Release key" events

"Press key" and "Release key" events are not supported on a Unified Comfort Panel.

Alarm control - Displaying logged alarms

Logged alarms are not displayed after starting the Runtime software if the static value "Logged alarms updated" is configured in the properties of the alarm control under "General > Alarm source". To make logged alarms visible in the alarm control, click the "Show logged alarms" or "Show and update logged alarms" button in the alarm control.

The "Update and display logged alarms" list shows a maximum of 100 alarms.

"StopRuntime" system function

The StopRuntime system function only works if encrypted transfer has been configured in the Runtime settings of the Panel.

System function "StartProgram" in the Scheduler

Calling up a program via the "StartProgram" system function in the Scheduler is not supported by Unified Comfort Panel. Instead, use a "Tags" trigger in the Scheduler to initiate the system function.

Access to MS SQL databases and SQLite databases

You can access MS SQL databases and SQLite databases via JavaScript functions from Unified Comfort Panel. Note that the resource utilization is directly proportional to the number of the requests to the database and the size of the data to be read or written. This means that the accesses to databases also affect the performance in Runtime. In order to not impair the operation as a visualization system the possibility to access databases should be used prudently.

From a Unified Comfort Panel you can only access databases which support the driver "Microsoft ODBC Driver for SQL Server" in Version 17.9.

These include:

- Azure SQL Database
- Azure Synapse Analytics
- Azure SQL Managed Instance
- SQL Server 2019
- SQL Server 2017
- SQL Server 2016

- SQL Server 2014
- SQL Server 2012

Date/time values

Column sorting of date/time values in the alarm control

In the engineering system, you can configure different output formats for displaying date/time values. The following applies to the column sorting of these values in an alarm control on the Unified Comfort Panel:

The following output formats do not support sorting in columns:

- {D,@EEE, MMM dd, `yy}
- {D,medium}
- {D,long}
- {D}

The following output formats support sorting in columns only in English:

- {D,@dd. MMMM yyyy}
- {D,@dd. MMM}
- {D, @MMM dd, yyyy}
- {D, @MMMM dd}
- {D} {T}

Display of date/time values in the alarm and trend control

After the first download of a project, date/time values in alarm and trend controls are shown in English time format by default. If your prioritized language is not English, then switch the language once to English and then back to your prioritized language. After that, the date/time values are shown in alarm and trend controls in your prioritized language.

Gauge output format

If you specify the value {H} (Hexadecimal) in the Inspector window of a gauge under "Properties > General > Scale > Output format", negative values in the gauge will not be displayed correctly. If you want to output negative values via a gauge, use {F}, {N} or {I} as output format.

Runtime Collaboration - function trend control

If a function trend control is configured in a screen to display logged data and you access this screen with Runtime Collaboration, no existing logged data is displayed in the trend control. The first value displayed is the value that will be written to the log after the screen is opened.

User management

In the engineering system you can configure a maximum session duration for a role as well as for a user. If these values are different, only the smaller of the two values is transferred to the Panel during loading.

Differences between simulation and Unified Comfort Panel

There are differences in the simulation regarding the following screen objects as compared to the Unified Comfort Panel Runtime:

- Alarm control:
  - Filter options
- Faceplates:
  - Scaling in JavaScript method "OpenFaceplateInPopup"
  - Arrangement of the faceplate containers for JavaScript method "OpenFaceplateInPopup"
- Screen window:
  - Scaling with system function "OpenScreenInPopup"
- I/O field: Default display for Word and DWord data types
- Parameter set display: Default display for Word data type
- Browser:
  - The display of pages of the S7 WebServer is not possible in the simulation.

Loading a project

When you have made changes to the Runtime settings in the User management area in Engineering, you must ensure before loading that the page "Security > UMAC settings" is closed in the Control Panel of the Unified Panel. Otherwise, the changes may not be applied after loading.

Simulation in connection with SD-X51

The simulation of tag retentivity and alarm logging in connection with the "SD-X51" storage medium is not supported.

2.11 Remote access to a Unified device

Contents

Information that could not be included in the online help and important information about product features.

Synchronize client values with server time

By default, the following controls display values with client time:

- Alarm control
- Trend companion
- Trend control
- f(x) trend control
- Process control

To synchronize the time displayed on the client with the server, proceed as follows:

- iOS devices:
  To prevent an iOS device from synchronizing with time.apple.com, create a profile file and upload it to the device.
  Profile files enable time synchronization within a secure corporate network.
- Android devices:
  Use a third-party app for time synchronization.

Access to a Unified device

Ensure you have the latest operating system and browser version if you want to access Runtime Unified with this device.

WinCC Unified displays the runtime elements in HTML5. The browser used also has to support this standard. Since the browsers interpret HTML5 differently, it is possible that objects are displayed differently depending on the browser and the browser version used.

Compatibility tests were performed for the following browsers. The focus of the compatibility tests was on the browsers marked with *:

Operating system	Browser
Microsoft Windows	• Google Chrome* • Microsoft Edge • Mozilla Firefox, Mozilla Firefox ESR
Android	• Google Chrome* • Firefox • Edge
iOS, Mac	• Safari* • Google Chrome • Firefox • Edge

With respect to the performance and support of the standard Runtime elements, Google Chrome has proven to be the preferred browser. Its memory requirements are slightly higher than those of the other browsers.

Note

Performance differences in different versions of individual browsers

The browser versions can differ from each other, which can result in different behavior regarding the memory requirements and speed.

Note

Suitability for continuous operation

MS Edge and Mozilla Firefox have proven to be problematic in continuous operation.

Unified Collaboration

Unified Collaboration is only permitted between devices with the same device version (starting from V16).

If Unified Collaboration uses local user management and the Collaboration partners are configured in different projects, it is possible to create users with the same name but different function rights. If one of these users logs on to a device in Runtime, the user has the function rights configured for this device as well as the function rights configured for the Collaboration partner. If you use several projects, you should configure the local user management identically in all projects.

2.12 Working with plant objects and plant views

Contents

Information that could not be included in the online help and important information about product features.

Introduction

You have the possibility to upgrade the project to V17 when opening your V16 project with a system configuration. The configured Runtime version of the HMI devices must be changed separately to V17.

Updating blocks

If blocks from a global or from a project library are used in the project, do the following:

1. Open the plant object type that uses a block from a library.
2. Delete the interconnection of the PLC tag.
3. Open the "Library" task card.

4. In the library, open the shortcut menu of the block used.
5. Click on "Project" under "Update types".
   The "Update types in the project" dialog opens.
6. Select the option "Force update (types including their dependent types are updated regardless of their version number)".
   All other options are already selected by default.
7. Click "OK".
8. Compile the PLC.
9. Interconnect the PLC tag for the plant object type.
10. Compile and download your project.

Note

Alternatively, delete all plant objects (instances) from the plant hierarchy, perform the update and create all plant objects (instances) again in the plant hierarchy. If you have linked a screen to the plant object in the "Visualization" tab, deleting the plant object not only deletes the link, but also entire screen. Create a copy of the screen and link the screen to the new plant object after updating the blocks.

2.13 Audit

Contents

Information that could not be included in the online help and important information about product features.

Restore database segments in Runtime

To analyze the logged and deferred data, as an HMI Administrator of an HMI device, you can restore backed up audit database segments in Runtime. After the analysis, these segments can be discarded again to empty the memory space.

WinCC Unified 3

3.1 Introduction

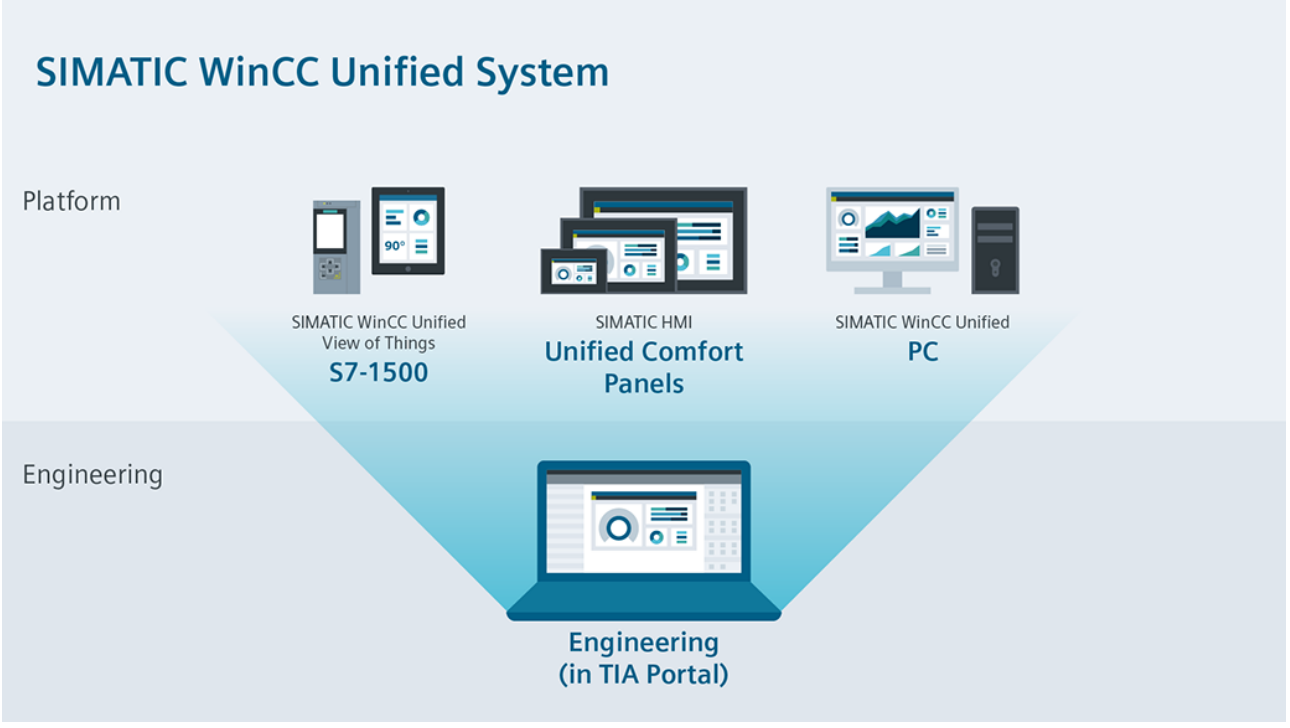
WinCC Unified helps you meet the challenges of digitization in machine and plant construction. Based on native web technologies such as HTML5, SVG and JavaScript, you can work independently of devices and environment.

An integrated system - from Panel to PC

For you, this means being able to efficiently configure a large number of mobile end devices on one engineering platform, such as Unified Comfort Panels, Unified PCs, Control Centers or applications in View of Things. In this way, WinCC Unified combines the advantages of strong products like WinCC Advanced, WinCC Professional and WinCC Comfort in one software and offers you maximum flexibility, scalability and efficiency.

Overview of advantages of the uniform engineering system in the TIA Portal:

- Reuse of components on all WinCC Unified platforms
- End-to-end usability of UI controls
- Uncomplicated device replacement between all devices that work with WinCC Unified



From the panel on the machine to the complex SCADA solution: WinCC Unified offers various options for industry-specific requirements and can be extended by user-specific applications through its open interfaces.

What makes WinCC Unified so special?

WinCC Unified combines SIMATIC HMI and SCADA in one product. This means WinCC Unified users can configure runtimes for device types across classes with one engineering system. An engineering system that is installed on a Unified PC, for example, can be used to configure runtimes for Unified Comfort Panels and Unified PCs. WinCC Unified therefore offers tremendous scalability and flexibility in the use of device classes within a system or via remote access for distributed systems.

3.2 Additional documentation

Siemens Industry Online Support will provide you with the latest documentation, which will help you with the initial working with and migrating to WinCC Unified.

Getting Started

A guideline to getting started quickly with WinCC Unified is available online: Getting Started (https://support.industry.siemens.com/cs/us/en/view/109801175).

Guideline for migrating to WinCC Unified

You can get support for migrating to WinCC Unified here: Guideline for migrating from Comfort Panel to Unified Comfort Panel and from WinCC Runtime Advanced to WinCC Unified PC Runtime (https://support.industry.siemens.com/cs/us/en/view/109768002).

Familiarize yourself with the differences between Comfort Panel/Runtime Advanced and Unified Comfort Panel/Unified PC Runtime. In addition, you will learn how you can migrate the essential elements (for example, pop-ups, slide-ins) to WinCC Unified.

Data2Unified

Continue to use the project content you have created when you switch to SIMATIC WinCC Unified and save engineering time with the "Data2Unified" add-in.

Information and download are available from Industry Online Support: Data2Unified (https://support.industry.siemens.com/cs/us/en/view/109770510)

The "Data2Unified" add-in is being continuously developed further (see the last change), so that even more TIA Portal project elements will be gradually supported.

The latest version of the add-in is provided for downloading in this contribution.

3.3 Creating a user interface efficiently

Screen editor

With the Unified screen editor, you can create user interfaces for various applications with a uniform structure. You will receive support particularly in the following areas:

- Operation and display in the screen editor, e.g.:
  - Labeling of screen objects for direct editing
  - Display of catch lines
  - Drawing of lines and polygons
- Drag-and-drop, e.g.:
  - An IO field is created during drag-and-drop of tags into the screen editor.
  - A parameter set control view is created during drag-and-drop of parameter set types.
- Changing properties of screen objects:
  - If multiple screen objects are selected, you can edit the properties at the same time.

Information on configuring screens can be found under "Basics of screens (Page 259)" and the following sections.

- Configuring screen objects (Page 361)
- Configuring text lists and graphics lists (Page 446)
- Configuring dynamization (Page 462)

Faceplates

Unified Faceplates are the new generation of HMI picture modules. Faceplates are user-defined groups of display and operating objects that can be reused as needed and thus reduce the configuration effort.

Faceplates are saved in a library and are managed there. The advantages of the type-instance concept are fully used. All functions that you are already familiar with from the library, such as updating of types, are supported.

You can create different versions of a faceplate type for use in screens or as a pop-up. Flexibly replace faceplate versions that are already in use with updated versions.

To connect the tags and interface properties defined in the faceplate individually to the process for each instance, use the different faceplate interfaces. For interface properties, use the pre-defined data types or the interface properties of the type "Configuration string" to assign the properties of each data type to a faceplate or the screen object in it using a script.

You can find more information under "Configuring faceplates (Page 524)".

Styles

WinCC Unified provides you with styles during the configuration to enable a user-friendly and flexible design of your runtime.

You can use these styles to easily customize the appearance of the runtime for each HMI device. While runtime is running, the style, for example, switches to a dark mode during a shift change.

You can find more information under "Using styles (Page 261)".

Custom web controls

Custom web controls are independent web pages with an interface to Unified Runtime. Custom web controls offer you the option of adding your own elements to the visualization elements provided. Custom web controls thus extend usability and functionality to achieve an optimal visualization result.

Custom web controls are run on the web client and hosted in Runtime Unified. A custom web control can be displayed as an independent web page in any web browser and on any mobile end device.

You can find more information under "Custom web controls (Page 7790)" and in the following sections.

Dynamic SVG graphics

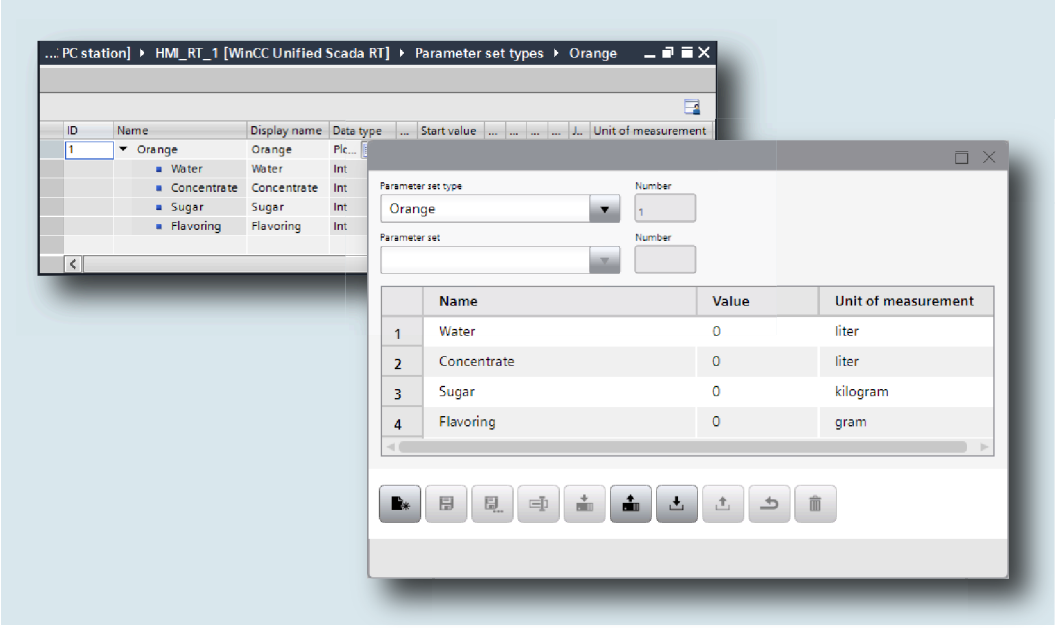
You can use dynamic SVG graphics in WinCC Unified. The Siemens Graphic Library offers a large selection of dynamic SVG graphics in the "Dynamic widgets". You use these graphics to design screens for your runtime that you can then configure as needed.

You can find more information under "Managing dynamic SVG graphics (Page 358)".

3.4 Controlling with parameter sets

A parameter set represents a recipe. Parameter sets are used for higher-level control.

Parameter sets are based on parameter set types which, in turn, are based on user data types. In this way you can achieve a high degree of reuse. Use the control "Parameter set control" to control the parameter set.



Parameter set types support the nesting of complex data types. You can, for example, assign another user data type to a user-data-type element as the data type.

You can find more information under "Configuring parameter sets (Page 863)".

3.5 Using distributed systems

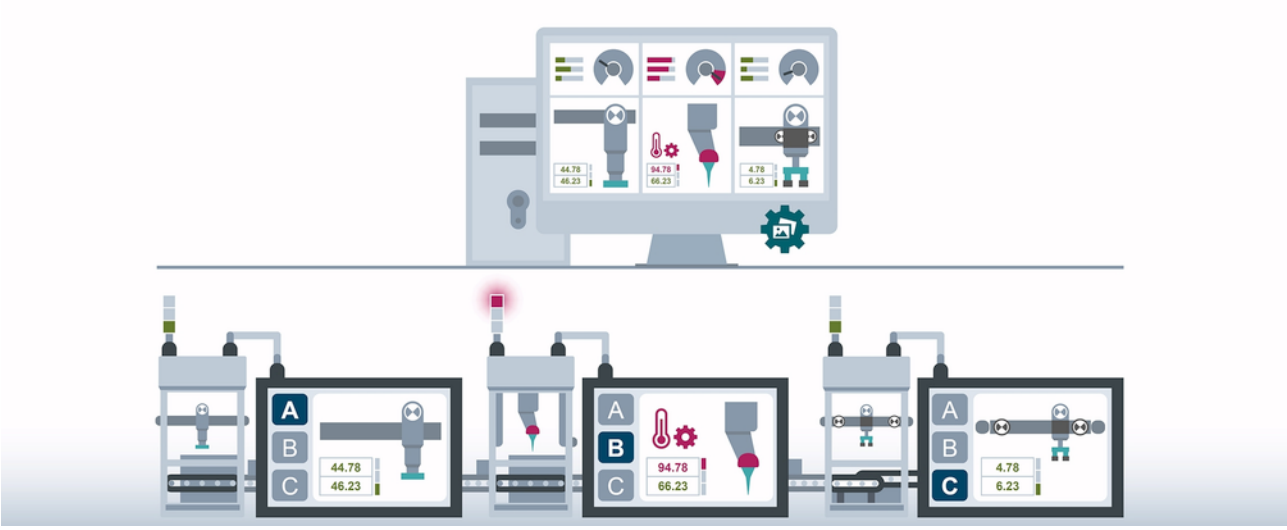
WinCC Unified offers remote access options.

Unified Collaboration allows the cross-device display and operation of individual screens.

With the web client, runtime access is possible from any device.

Unified Collaboration

With Unified Collaboration you can avoid redundant configuring. You have the option to create an overview with content from different Unified HMI devices. To this end, you access screens of a different HMI device across devices. You can display and operate screens of a different HMI device. Screen windows and alarms are supported here.



An HMI device that participates in collaboration is referred to as a collaboration device.

Unified Collaboration is supported by Unified PC and Unified Comfort Panel and allows cross-project access.

You can find more information under "Unified Collaboration (Page 7557)".

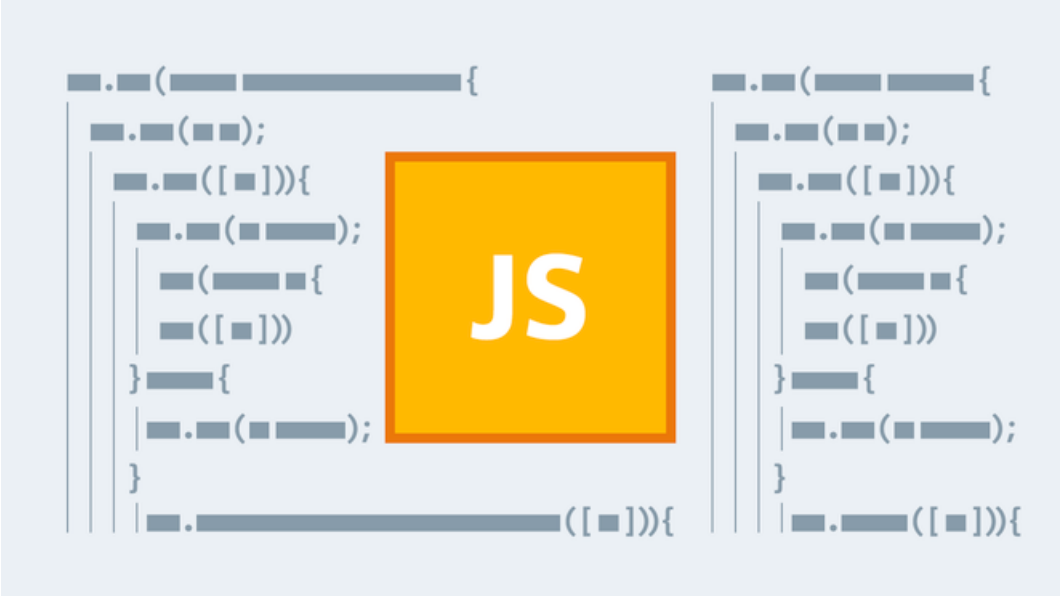
Web client

You use the web client to access a Unified PC or a Unified Comfort Panel via remote access. Access is possible from any device. The web client is used as standard remote access for Unified PC. The clients are independent of one another and independent of the locally displayed runtime. This means the runtime can be operated by multiple users simultaneously. To prevent overlaps and undesirable effects, we distinguish between synchronous and asynchronous functions.

You can find more information under "Web Client (Page 7589)".

3.6 Dynamization and automation through scripts

WinCC Unified provides a modern scripting environment that you can use to automate system components, such as the graphical runtime system. The scripting environment maps individual elements of the system components, such as the screens of the graphical runtime system, via the object model. The object model supports you in solving different tasks through runtime scripting and in controlling processes.



The scripting environment offers:

- Efficiency and the latest technologies
  The scripting environment supports Unicode and uses JavaScript (JS) as the scripting language. The scripting environment is object-oriented and offers asynchronous operations for high-performance and secure script execution.
- Support of mass data
  The scripting environment is optimized for the processing of mass data, for example, writing 1000 tags in one pass. Special script objects are available to this purpose that handle numerous HMI objects of the same type. These script objects execute operations on all the HMI objects simultaneously instead of processing each HMI object individually.
- Input support:
  - Syntax highlighting
  - Snippets (code templates)
  - Referencing HMI objects
  - Tooltips
  - Autocomplete
  - Error marking and correction
  - System functions (Page 909)

You can find more information under "Introduction to runtime scripting (Page 969)" and in the following sections.

Here you will find Notes on creating scripts (Page 972), a description of the Script editor (Page 981) and examples of scripts (Page 989).

In the SiePortal, your can find support for creating scripts in JavaScript: Tips and tricks for creating scripts (https://support.industry.siemens.com/cs/ww/en/view/109758536).

3.7 Central user management

The user management of WinCC Unified allows for the plant-wide, central management of users including optional connection of Microsoft Active Directories. The user management forms the basis for the efficient and integrated management of personalized access rights in a plant. Security risks are significantly reduced. The person-specific assignment of roles and rights minimizes the maintenance effort. At the same time, a high level of transparency is achieved.

The User Management Component (UMC) option allows for the setup of a project-wide, central user management.

The user management of WinCC Unified offers:

- Central, cross-project management of user groups and users in a system.
- Import of user groups and users from Microsoft Active Directory.
- Failsafe performance thanks to redundant design of a User Management Control domain (UMC domain).
- Load distribution thanks to multiple UMC stations in one UMC domain.

You can find more information under "User management in the TIA Portal (Page 6887)" and in the following sections.

3.8 Connectivity

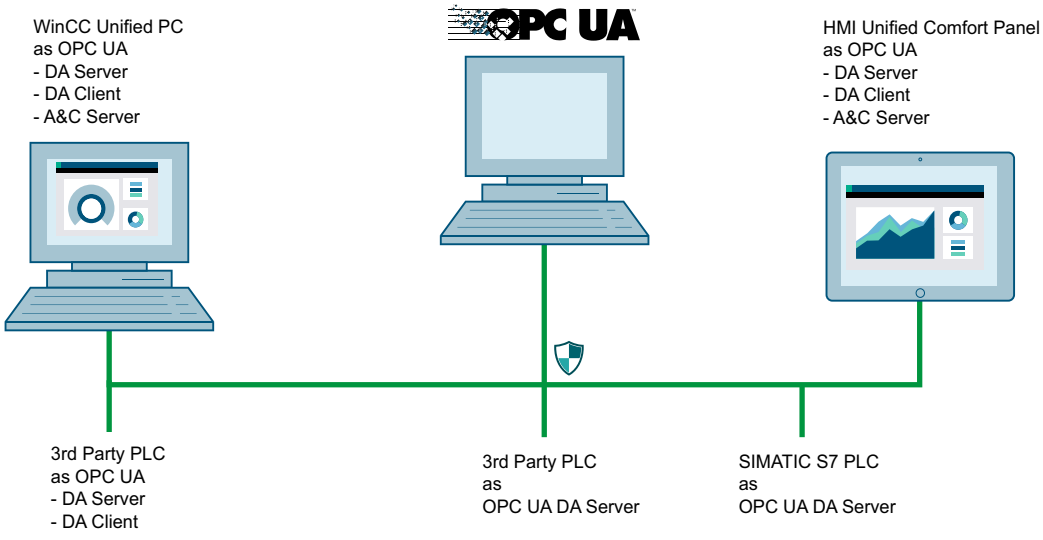
WinCC Unified offers you the highest level of integration of SIMATIC PLCs. The TIA Portal of WinCC Unified offers you an easy connection of

- Up to 128 PLCs for Unified PC systems
  If more than 10 connections are used, SIMATIC NET PC software must be installed.
- Up to 16 PLCs for Unified Comfort Panel systems
- S7-1500 Software Controller
- S7-1200/1500
- S7-300/400
- Support of native 3rd party products via
  - Modbus TCP
  - Allen-Bradley EtherNet/IP
  - Mitsubishi
  - Omron
  - Support of additional products via the Channel Support Package

You can find more information under "Basics of communication (Page 6973)" and in the following sections.

OPC UA

OPC UA is a standardized manufacturer-independent software interface for data exchange in automation engineering and is the technology succeeding OPC. OPC UA is platform-independent and supports different protocols as communication medium.



With WinCC Unified you can use OPC UA as connection standard for WinCC Unified PC and Unified Comfort Panels.

You can use WinCC Unified PCs and Unified Comfort Panels as needed:

- OPC UA DA server
- OPC UA DA client
- OPC UA A&C server

The WinCC Unified security concept guarantees your data security through secure OT/IT connections via OPC UA for connections to 3rd party applications.

You can find more information under "WinCC Unified OPC UA server (Page 7025)" and "WinCC Unified OPC UA client (Page 7044)".

3.9 Logging and traceability

Traceability and therefore the documentation of production data is becoming increasingly important in many sectors such as the pharmaceutical industry, the food and beverage industry, and the related mechanical engineering industry.

Logging of production data in electronic form offers many advantages compared to paper documents, such as simple acquisition and logging of data. WinCC Unified supports the two database types Microsoft SQL and SQLite.

However, it is also important to ensure that data cannot be falsified and that it can be read at any time.

Therefore, sector-specific and cross-industry standards have been developed for the electronic documentation of production data. The most important set of regulations is the FDA Guideline 21 CFR Part 11 for electronic data records and electronic signatures issued by the FDA, the US Food and Drug Administration.

In addition, different EU regulations apply, such as EU 178/2002, depending on the industry. Requirements for production systems in these industries have been developed based on 21 CFR Part 11 and the corresponding interpretation to comply with GMP (Good Manufacturing Practice). They are also required for other industries.

With WinCC Unified, you can configure projects in compliance with GMP and thus ensure traceability and data integrity.

WinCC Unified provides you with the optimal logging solution for each of your applications. You can start your application with file-based logging and extend it by a database option. File-based logging is especially suited for small and medium applications.

You can find more information under "Log basics (Page 837)" and in the following sections.

Detailed descriptions can be found at "Logging alarms (Page 769)" and "Logging tags (Page 653)".

3.10 Configuring plant hierarchies

Plant Model

WinCC Unified offers you object-oriented configuration. With the plant model, you can define reusable plant object types by arranging the associated plant object instances in hierarchical plant views.

In this way, you can model the technological hierarchy of your machine or unit/plant, for example, based on user-defined or standardized technology objects.

You create the plant structure from individual objects, each of which represents a component or a plant unit. Each object is configured in the context of the operator control and monitoring solution.

In plant object types, you combine all required configuration elements for visualization, e.g. faceplates, tags, alarms, scripts. Changes to the plant object type automatically affect all instances. This translates into significant time savings, especially for plants with a high degree of standardization.

If necessary, you can start object-oriented plant modeling based on the engineering data and derive the configuration of the HMI devices and automation systems from this.

Break the machine or unit/plant up into reusable technological units and arrange them hierarchically in a technological plant view according to the plant structure.

The following options are available to you in technology-oriented and object-oriented configuration:

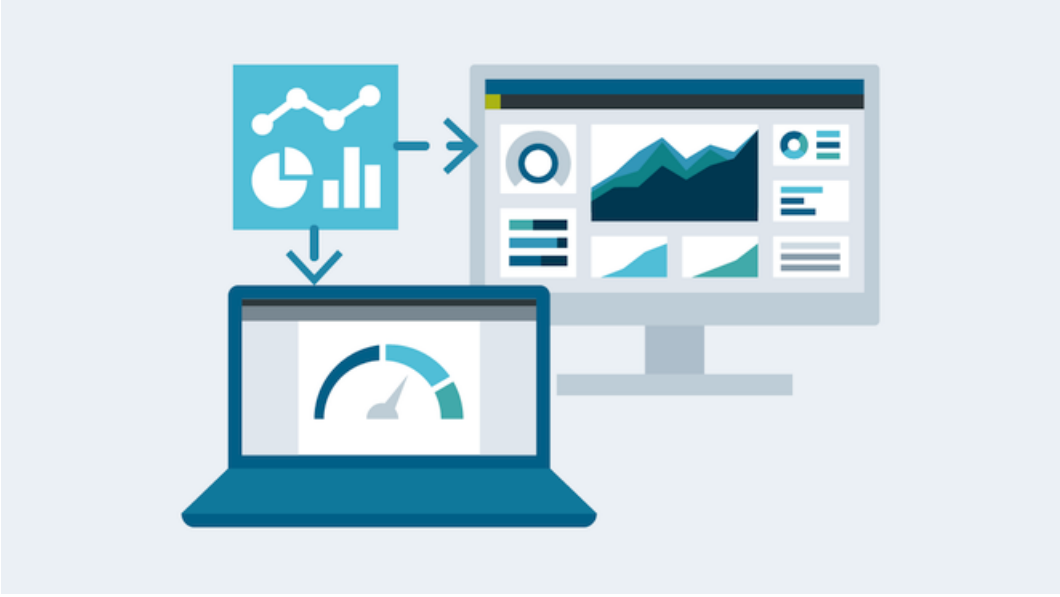
- Creating various hierarchical plant views: technological view, building view, independent of the HMI device that is used.
- Configuration of plant objects and plant object types with data elements for mapping the actual plant configuration
- Access to plant objects (data elements, HMI alarms, logs, screens, etc.)
- Generation of the screen hierarchy
- Expansion of configured plant objects and types using Plant Intelligence options

You can find more information under "Visualizing plant objects in runtime (Page 7113)" and in the following sections.

Plant Intelligence options

The Plant Intelligence options supplement the WinCC Unified visualization system with efficient functions:

- Calendar: For the visualization of structured planning of the production processes.
  You can find more information under "Calendar Basics" and in the following sections.
- Performance Insight: For the calculation of individual Key Performance Indicators (KPIs) according to ISO standard 22400 and comprehensive selection of WinCC controls for their display and analysis.



You can find more information under "Basics of Performance Insight" and in the following sections.

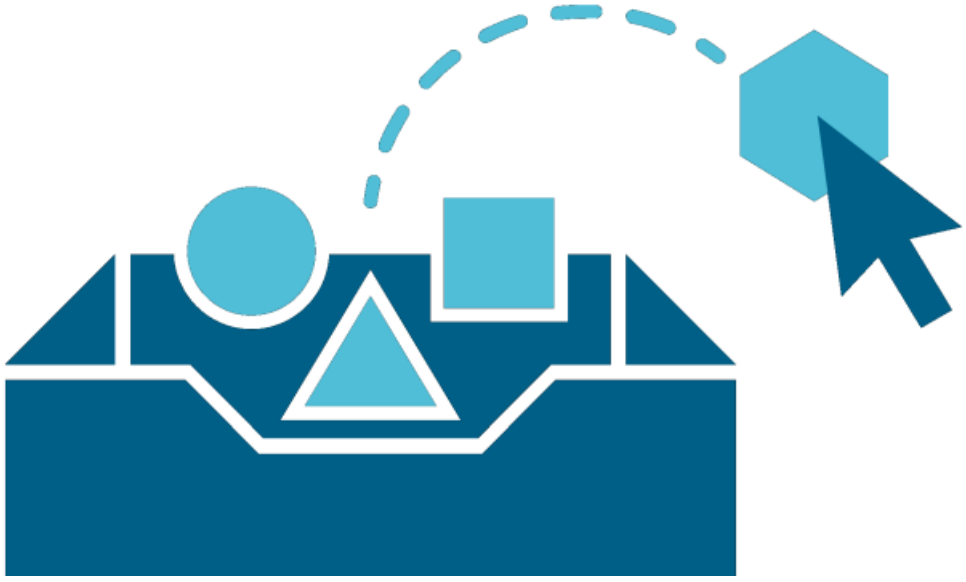
- Line Coordination: Automation of recipe-controlled and batch-controlled production processes.
  You can find more information under "AUTOHOTSPOT" and in the following sections.
- Sequence: Fast and easy change of production processes and parameters without changing the PLC program.
  You can find more information under "AUTOHOTSPOT" and in the following sections.

3.11 Working with libraries

3.11.1 Re-using libraries

The object-oriented HMI concept of WinCC Unified allows you a high degree of reusability of elements both within and across projects. The new library concept contains the following elements that you can configure and reuse in your projects as needed:

- HMI Faceplates
- HMI user data types
- HMI styles
- HMI style sheets
- Graphics and dynamic SVG graphics
- Script modules



Types and instances

You can create instances from types that you have created in a library and which you manage and edit there. Use them in your project. The instances are linked to the respective type.

Master copies and copies

In addition, you can access many master copies for cross-project configurations in the global libraries. You can also yourself create master copies and manage them in the project library or in a global library that you have yourself created.

From a master copy, you can create an independent copy in the project, which you can edit independently of the underlying library object.

3.11.2 Basics on libraries

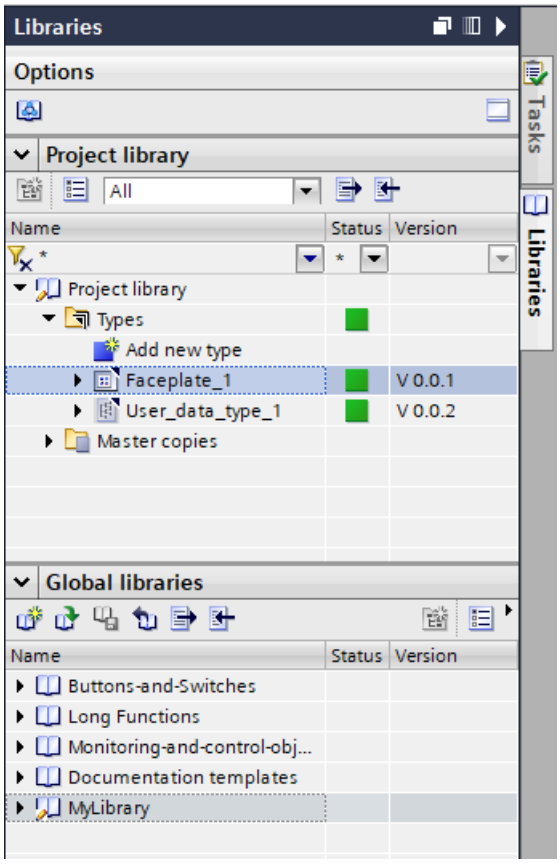
Introduction

Store all objects you need frequently in the libraries. An object that is stored in the library only has to be configured once. It can then be used repeatedly as often as required. Library objects extend the number of available screen objects and increase the effectiveness during configuration through multiple use.

Your WinCC software package is supplied with comprehensive libraries that contain, for example, "Motor" or "Valve" objects. You can also define library objects yourself.

Libraries are managed in the "Libraries" task card or in the library management. The following libraries are available:

- Project library
- Global libraries



Project library

There is one library for each project. Objects of the project library are stored alongside with the project data and are available only for the project in which the library was created. If the project is moved to another PC, any project library created in it is also moved.

To use the library object of the project library in other objects, move or copy the object into a global library.

Global libraries

A global library is saved independently of the project data in its own file with the extension ".alxx", whereby "xx" stands for the current WinCC version number.

A project can access several global libraries. A global library may be used concurrently in several projects.

When a library object is changed by a project, this library will be changed in all projects in which these libraries are open.

Library objects

A library can contain various WinCC objects.

Examples of types:

- HMI Faceplates
  If you want to use configurable object groups several times in screens and change them centrally, create faceplates for them. If you change the properties of a faceplate in the library, the changes affect all screens and scripts that use this faceplate.
- HMI user data types
- HMI styles
- HMI style sheets
- Graphics and dynamic SVG graphics
- Script modules

Examples of master copies:

- Complete HMI device
- Screen
- Tag
- Parameter set type
- Script

3.11.3 Types and master copies

Introduction

Both the "Project library" and the "Global library" contain the two folders "Master copies" and "Types". You can create or use the library objects either as a master copy or a type.

Types

Create instances of objects of the "Types" folder and use these in your project. The instances are bound to their respective type.

More information is available here (Page 176).

Master copies

Use master copies to create independent copies of a library object.

More information is available here (Page 185).

Administration of the library objects

You can copy and move library objects to other libraries. You copy master copies to the "Master copies" folder or any subfolder of "Master copies". You can only insert types in the "Types" folder or any sub-folder of "Types".

3.11.4 Creating types and master copies

Creating a new type

You can create a new type in the project library.

You can later copy the type into a global library that you yourself have created.

1. To create a new type, select "Add new type" in the project library.
   The "Add new type" dialog is displayed.
2. Select the class of the type.
3. Provide further information on usage.

- or -

1. Drag and drop an object from the project tree to the "Types" folder in the project library.
   The object must be suitable for creating a type, for example, a user data type.

More information: Using types and their versions (Page 176)

Create a new master copy

1. To create a master copy, drag and drop an object from the project tree to the "Master copies" folder of the project library or a self-created global library.
   The object must be suitable for creating a type. You can use objects that you yourself have created in the project tree as a master copy.
   The cursor indicates if creating the master copy is possible.

More information: Using master copies (Page 185)

3.11.5 Managing libraries

3.11.5.1 Overview of the library management

Function of the library management

Master copies and types with dependencies on other library elements are subject to functional restrictions. For example, they cannot be deleted while dependencies still exist. This prevents other library elements from becoming useless. The library management is used to identify the dependencies and to create an overview of the work progress.

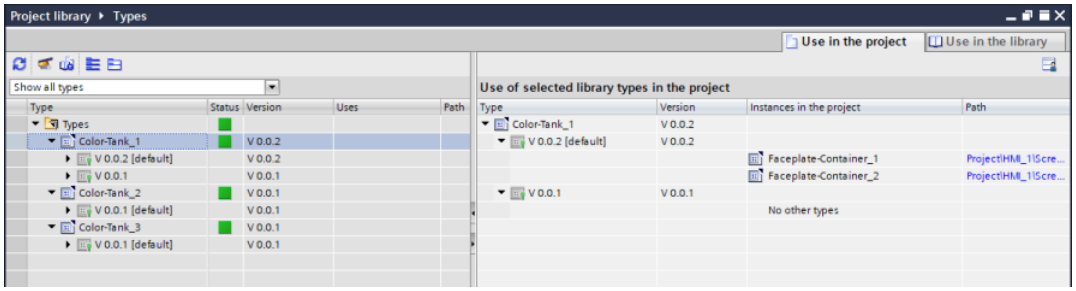
The library management offers the following functions:

- Display of the correlations of types and master copies
  If a type is referenced in other types or master copies, the relationships are displayed in the library management. You will also be able to see which library elements reference a type or a master copy.
- Display of points of use of types in the project
- Display all types that include a version with the "In test" or "In progress" status

Layout of the library management

The figure shows the library management, which consists of the following components:

- Toolbar of the library management
- "Types" area
- "Use" area



Toolbar of the library management

You can perform the following tasks in the toolbar of the library management:

- Update view
  If the project was changed, you can update the view of the library management.
- Clean up library
  You can clean up the project library and global libraries. Cleaning up a library deletes all types and type versions that are not linked to any instance in the project.
- Harmonize project
  By harmonizing a project, you adapt the names and the path structures of type uses in the project to the corresponding names and path structures of the types within a library.
- Collapse all
  Shows only the top node
- Expand all
  Shows all nodes (types and versions)

"Types" area

The "Types" area displays the contents of the folder or type you selected in the "Libraries" task card.

The selection list following filters:

- Show all types
- Types with pending changes
- Released types
- Types with multiple versions
- Types not used in the project
- Types with inconsistencies in the default version
- Highest version of the type without "default" identifier.

For each type, the types that it references are displayed.

"Uses" area

The "Uses" area gives you an overview of the points of use of the selected types. The "Uses" area is divided into two tabs:

- "Uses in the project" tab
  In the "Uses in the project" tab, display the instances of type versions and the path to the point of use in the project.
  To get to the instance in the project, click the entry under "Path".
- "Uses in the library" tab
  The "Uses in the library" tab is used to show all points within the library at which a type is used.

3.11.5.2 Opening library management

Procedure

To open the library management, follow these steps:

1. Open the "Libraries" task card.
2. Select a type or any folder that contains types.
3. Select the "Open library management" button  .
   - or -
4. In the shortcut menu, select "Library management element".

Result

The library management opens and the types are displayed with their versions.

Working with large or multiple monitors

The library management window can be detached and moved as desired.

3.11.5.3 Filtering types in the library management

Introduction

A filter function in the library management enables you to limit the displayed types. The following filters are available:

- Show all types
- Types with pending changes
- Released types
- Types with multiple versions
- Types not used in the project
- Types with inconsistencies in the default version
- Highest version of the type without "default" identifier.

Requirement

At least one type has been created.

Filtering by types with pending changes

1. Select the "Types" folder in the project library.
2. Open the library management.
3. Select "Types with pending changes" in the drop-down list.
   The "Types" area only displays types that have the "in progress" status.

Filter by released types

1. Select the "Types" folder in the project library.
2. Open the library management.
3. Select "Released types" in the drop-down list.
   The "Types" area only displays types that have released versions.

Filtering for types with multiple versions

1. Select the "Types" folder in the project library.
2. Open the library management.
3. Select "Types with multiple versions" in the drop-down list.
   The "Types" area only displays types that have more than one version.

Filtering for types that have no instances in the project

1. Select the "Types" folder in the project library.
2. Open the library management.
3. Select "Types not used in project" in the drop-down list.
   The "Types" area only displays types that have no instances in the project.

Filtering by types with inconsistencies in the default version

1. Select the "Types" folder in the project library.
2. Open the library management.
3. Select "Types with inconsistencies in the default version" in the drop-down list.
   Only the types whose default version is inconsistent are displayed in the "Types" area.

Filter by types whose highest version is not the default version

1. Select the "Types" folder in the project library.
2. Open the library management.
3. In the drop-down list, select "Highest version of the type without default identifier".
   In the "Types" area, only those types are displayed whose highest version is not the default version.

3.11.5.4 Creating a global library

Introduction

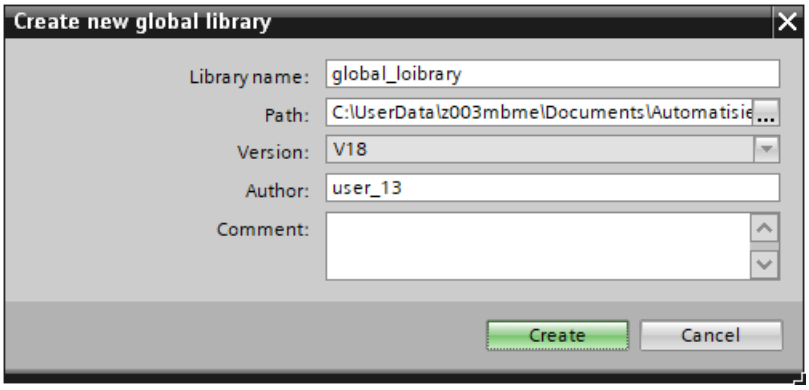
In the libraries you store the configured objects that you want to use several times in your configuration. To use objects in several projects, create a global library.

Requirement

- A project is open.
- The "Libraries" task card is opened.

Procedure

1. Click the icon under "Global library".
   The "Create new global library" dialog opens.



2. Enter a name.
3. Select the path where the new library is to be stored.
4. Click "Create".

Result

The new library is shown in the "Global libraries" palette. The global library contains the "Types", "Master copies" and "Common data" folders. Under "Common data" you can reports for the global library.

A folder with the name of the global library is created in the file system at the storage location of the global library. This actual library file is given the file name extension ".alxx", whereby "xx" stands for the current WinCC version number.

3.11.5.5 Saving a Shared Library

Introduction

A global library is saved as a separate file. This file contains all the objects of the global library, including the referenced objects. For example, the reference of a tag which was configured on an I/O field is also saved in the library.

WinCC prompts you to save the global libraries when you close WinCC or your project without saving. You also can store the global library during configuration, without storing the entire project.

Requirement

- A project with at least one global library is open.
- The "Libraries" task card is opened.
- A global library has been changed.

Procedure

1. Select the global library that you want to save.
2. Click the  icon in the "Global library" palette.
   You can alternatively select the "Save Library" command in the shortcut menu.

Save as:

1. To save the global library to another folder, select "Save as" from the shortcut menu.
2. Select the path in which you want to store the new library and enter a file name.

Result

The global library is saved under the current file name or the newly assigned file name.

3.11.5.6 Opening a global library

Introduction

In WinCC, the global libraries are stored in separate files. You can use a global library in every project.

Requirement

- You have saved a global library.
- A project is open.
- The "Libraries" task card is opened.

Procedure

1. Click the  icon in the "Global library" palette.
   The "Open global library" dialog box is displayed.
2. Select the path in which the library is stored.
3. Click "Open".

Note

To have the access to a global library from multiple projects, open the global library in write-protected mode. When a global library is read-only, access from other projects is blocked.

Result

The global library is displayed in the "Global libraries" palette.

3.11.5.7 Showing logs of global libraries

Logs listing all changes made to the global library are created when global libraries are updated. The logs are stored together with the global library and are always available once you have opened the global library.

Procedure

To open the logs of a global library, follow these steps:

1. Open the global library in the "Libraries" task card.
2. Open "Common data > Logs" in the lower-level folder.
3. Double-click on a log.
   The log is opened in the editor.

3.11.5.8 Updating a project with the contents of a project library

Introduction

After you have edited several types in the project library, update all instances in the project to the most recent version of the types from the project library.

Each of the following elements can be selected as source for the update:

- Individual folders within a library
- Individual types

Requirement

The "Libraries" task card or the library management is opened.

Procedure

1. Select a folder within the project library or individual types.
2. Select "Update types > Project..." from the shortcut menu.
   A dialog opens.
3. Select either the entire project or individual devices for the update.
4. Select "Update instances in project".
5. To delete all older versions of the updated types from the project library, select the check box "Delete unused type versions without "default" identifier from the library".
6. Confirm with "OK".

Result

All instances of the types are updated in the project to the most recent version of the selected types in the project library.

You can find a log of the update process in the project tree under "Common data".

3.11.5.9 Updating a library with the contents of another library

The following options are available for updating libraries:

- Updating a global library with types from another global library
- Updating the project library with types from a global library

Each of the following elements can be selected as source for the update:

- Individual folders within a library
- Individual types

Requirement

- The "Libraries" task card or the library management is opened.

Procedure

To update a library with the contents of a different library, follow these steps:

1. Select a folder within the library or individual types.
2. Right-click the source and select the "Update types > Library..." command from the shortcut menu.
   The "Update library" dialog box opens.
3. Select the type of library you want to update:
   - Select "Update project library" to update the project library with types from a global library.
   - Select "Update global library" if you want to update a global library.
4. Optional: Select the global library you want to update from the drop-down list.
5. Enable the desired update options:
   - Updating instances in the project
   - Delete unused type versions without "Default" label from the library
   - Force update
     Types are updated including their dependent types regardless of their version number.
6. Confirm with "OK".

Result

- Types not yet available in the target library are supplemented there with all their versions. More recent versions are added to the types that already exist in the target library.
  If a more recent version of a type already exists in the target library, the latest version is nevertheless copied from the source library and automatically assigned a newer version number.
- A log listing all performed changes to the target library is created for the update process.
  If you have updated the project library, you can find the log in the project tree under "Common data > Logs".
  If you have updated a global library, you can find the log in the "Common data > Logs" folder in the level below the global library.

3.11.5.10 Exporting and importing library texts

Introduction

You can export the texts of the library objects to an .xlsx file to edit them in MS Excel, for example, or export them for compilation.

You export and import texts of the following objects in the library:

- Individual library types and master copies
- Multiple library types and master copies
- All library objects of the project library or a global library

After editing or external compilation, you import the texts into the TIA Portal.

When the texts are imported, all texts from the import file are imported for the entire library, even if you have only selected one library object. The target languages of the import file must be activated in the project.

During import to a master copy, the texts of the template are overwritten in the library with the new texts from the import file. During import of the texts to a library type, the latest version is overwritten in the library with the new texts from the import file. If a version of a type has not been released yet in a project library, no texts can be imported for the entire project library.

Defining the source language and target language

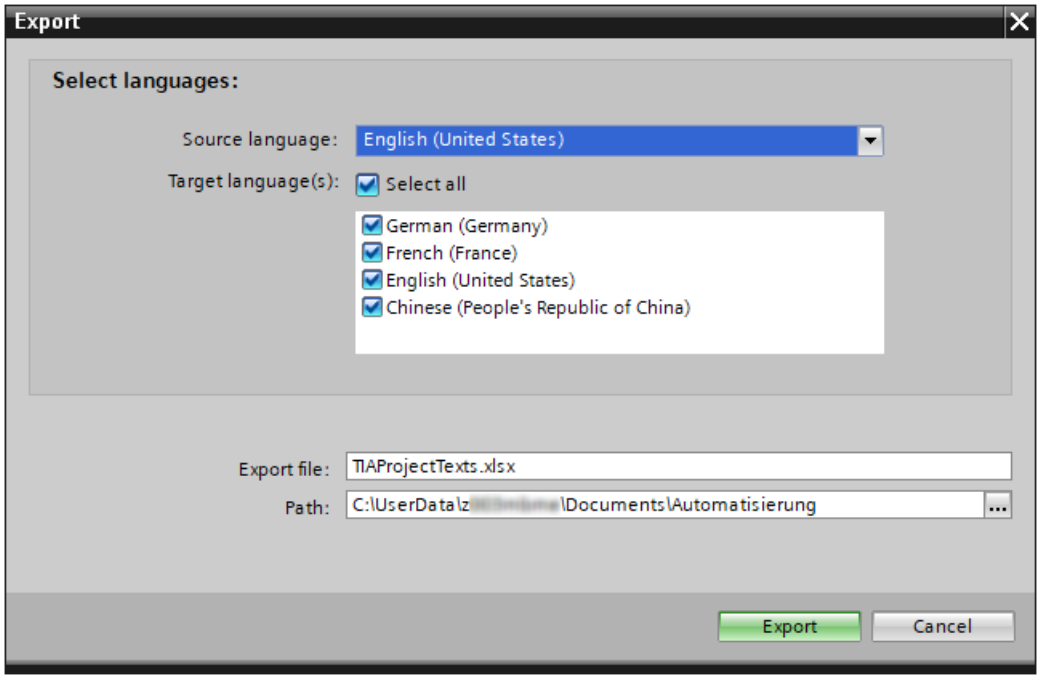
You define the source language and target language for export of the texts in the Export dialog. The selection of available languages depends on the project languages defined.

With the global library, the selection of available source and target language depends on the languages defined by the creator of the library. To see the available languages of the global library, double-click the entry "Library languages" in the project folder "Language and Resources" of the library in question.

Exporting texts

To export the texts of a single or several library objects, follow these steps:

1. Open the project library or a global library.
2. Select the library object in the library.
3. Select the command "Export library texts" in the shortcut menu of the object.
   Alternatively, click the "Export library texts" button in the toolbar.
   The "Export" dialog box opens.
4. Select the source language and the target language for export in the dialog.
5. Enter the name and path for the export file.



6. Click "Export".
   After successful export, the export file is stored under the specified path.

Importing texts

Note

Restrictions for text import

The texts which belong to the following library objects cannot be imported:

- Type instances which are contained in a master copy
- Library types whose versions have not yet been released and which have the status "In progress" or "In test"
- Write-protected global library

If importing into a project library, all versions must be released in this project library.

To import the texts after editing or compilation into the TIA Portal again, follow these steps:

1. Open the project library or the global library.
2. Select the command "Import library texts" in the shortcut menu of the object.
   Alternatively, click the "Import library texts" button in the toolbar.
   The "Import" dialog box opens.
3. Select the path and the name of the import file from the "Select file for import" field.
   Activate the "Import source language" check box if you have made changes to the source language in the export file and would like to overwrite the entries in the project with the changes.
4. Click on "Import".

3.11.6 Managing objects in a library

3.11.6.1 Displaying library objects

Introduction

The elements of a library can be displayed in the folder structure under the library or in the "Elements" palette.

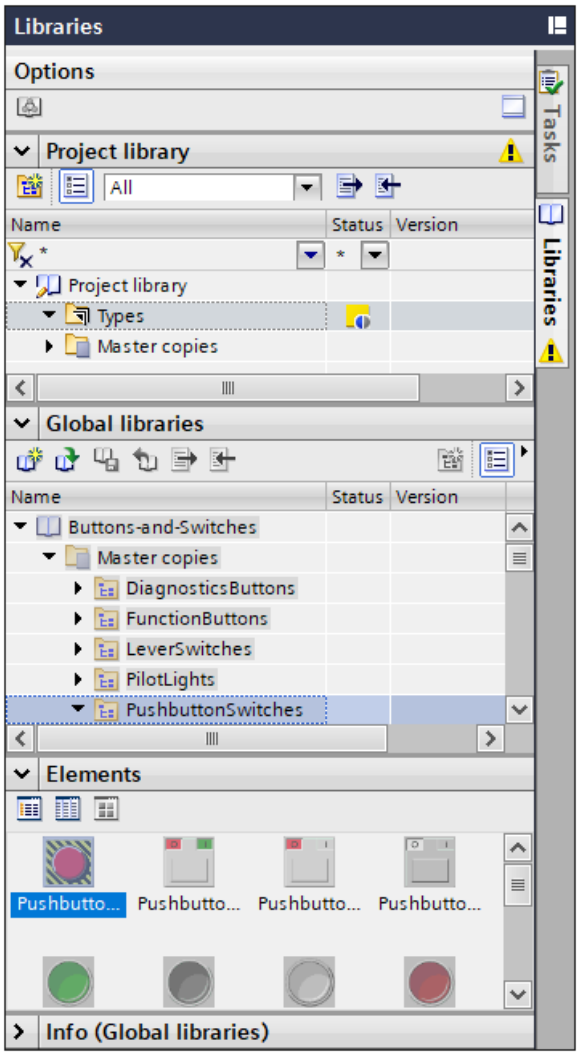
Requirement

- At least one library object has been created in a library.
- The "Libraries" task card is opened.

Displaying parts of the library objects

1. Select the folder of a library whose elements are to be displayed.
2. Click .
   The objects contained are displayed in the "Elements" palette.

3. To display components of an element, click on an element in the "Elements" palette.
   The parts of the element are displayed in the "Elements" palette.



4. Select a view:

Icon	View
	Details mode
	List mode
	Overview with icons

Result

The library objects are displayed with the selected view in the "Elements" palette.

3.11.6.2 Storing an object as master copy

Introduction

In libraries you can store WinCC objects, e.g. screens, tags, alarms, scripts or parameter set types as master copies. To do this, drag and drop the object from the project tree, the work area or the detail view into the library. If you have created subfolders in the library, you can also insert an object directly there.

Requirement

- The project tree or the work area is open.
- An object has been created.
- The "Libraries" task card is displayed.

Procedure

1. Select the object in the project tree or in the work area.

Note

Objects from the project tree as master copy

You can use all objects as master copy, which you can create new in the project tree:

- HMI devices, PLCs, technology objects, ...
- Screens
- Tag tables
- Parameter set types
- Scripts

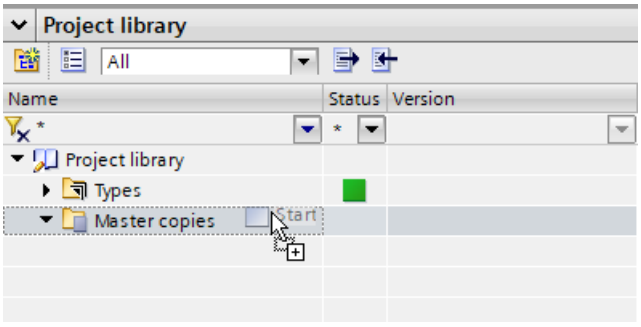
Note

Objects from a work area as master copy

You can use objects as master copy, which you can create new in a work area:

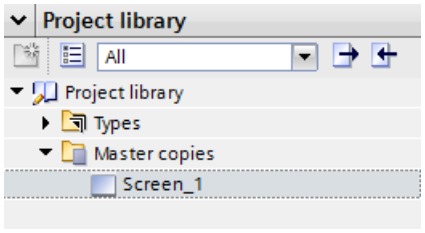
- Tags
- HMI alarms
- Tasks
- Cycles
- Text and graphic lists

2. Drag and drop the object into the "Master copies" folder in the library.
The mouse pointer is transformed into a crosshair with an appended object icon.



Result

The object is saved to the library for further use in multiple instances of your configuration.



3.11.6.3 Inserting a library object

Introduction

The system always assigns the inserted library object a name which consists of the name of the object type and of a consecutive number.

If the inserted object already exists, you have the option of replacing the object or of saving it under a new name.

You cannot insert library objects that are not supported by the HMI device.

Note

If you insert a screen with interconnected template from the library, the template will also be inserted. An existing, suitable template is not used.

Requirement

- The "Libraries" task card is opened.
- The editor in which you want to insert the library object is open.

Procedure

1. Select a library object from the library.
2. Drag-and-drop the library object to the position in the work area where you want to insert the object.
   The library object is inserted.

Result

If the object was contained in the "Copy templates" folder, you have inserted an independent copy of the library object in the editor.

If the object was contained in the "Types" folder, you have inserted an instance of the library object in the editor.

3.11.7 Using types and their versions

3.11.7.1 Status of versions of a type

Introduction

Depending on the point of use, the version of a type has different states.

Released version

The "Released version" status is available for all types, regardless of the point of use.

If you want to edit a released version, you must first create a new test version or an "in progress" version.

Released type versions of scripts and screens can be opened and viewed at their instance.

"In progress" version

When you create a new type or a new version of a released type, the type is set to the "In progress" state.

Types with the "in progress" state can be edited in the library management without the need for a reference to an instance in the project. Upon release, the compatibility of the type is tested by a consistency check.

"In test" version

Versions of HMI user data types can be in the "in test" state.

If you create a new PLC data type and add it as HMI user data type to a library, then this type is set to the "In test" state during editing. Select a device as the test environment before opening the editor for the type.

A version with "In test" status is linked to an instance in the project. You can set only one version to "In test" for each type at a given time.

An "In test" version may only be linked to a single instance in the project. Therefore, it is not possible to copy an instance to the clipboard, to duplicate it or to create an additional type from the instance as long as it has "In test" status.

3.11.7.2 Adding types to a project library

Requirement

- A project is open.
- An HMI device has been created and opened.
- The project tree is open.
- The "Libraries" task card or the library management is opened.

Procedure

1. In the "Project Library" palette, select a folder that contains types
   - or -
   Open a folder from the project library that contains types in the library management.
2. Select "Add new type..." from the shortcut menu.
   - or -
   Click "Add new type" in the "Project library" palette.
   The "Add new type" dialog is displayed.

3. Select the type.
4. Specify the device for which the type is being created.
5. Click "OK".
   Depending on the selected type, the editor for editing the type opens.
6. Close the note at the top of the window and edit the type.
7. Release the version of the type after the editing.

Result

You have added a type to the project library.

3.11.7.3 Create a new version of a type

If you create a new version of a type, the point of use of the type determines the status of the newly created version.

Requirement

- The "Libraries" task card is opened.
- A type has been created and released.

Procedure

1. Select the released type.
2. Select "Edit type" from the shortcut menu.

Result

- A new version of the type is created.
- The version of a type has the state "In progress".
  - or -
  The version of an HMI user data type has the state "In test".
- The editor opens.

3.11.7.4 Editing a type

Introduction

To edit a type, open the type in the library management or in the "Project library" task card.

The editor for the type opens. In the note at the top of the editor you can find information about the status of the type as well as about additional options for editing.

Requirement

At least one type is created in the project library.

Procedure

1. Select "Open" in the shortcut menu of the type.
   For types of the "User data type" type, the dialog for setting the test environment is displayed.
   The editor for the type is displayed.
2. Close the note at the top.
3. Edit the type.
4. To release the version of the type or to discard the changes, reopen the note at the top.
   

3.11.7.5 Consistency status of types

Types may no longer be consistent after changes. The "Status" column in libraries indicates whether a type is consistent or inconsistent. The following statuses are displayed:

Icon	Meaning
	The type is consistent.
	The type has more than one inconsistency.
	The default version of the type does not use the default version of its dependent type.
	The type has duplicate versions.
	More than one version of the type is instantiated in the device.
	A version other than the default version of the type is instantiated in the device.

3.11.7.6 Generating a faceplate as a type

Introduction

To define a new faceplate type, add a new type in the project library.

Procedure

1. Open the "Libraries" task card.
2. Select the "Add new type" command under "Types" in the shortcut menu of the project library.
   A dialog opens.
3. Select "HMI faceplate".
4. Specify the device for the new type.
5. Assign a descriptive name to the new faceplate type.

Result

The new faceplate type is created and displayed under the selected name in the project library.

The faceplate type is assigned the status "In progress" and the version 0.0.1.

The editor for a faceplate opens.

Using a faceplate type

- Open a screen to edit.
- Drag and drop the type from the "Libraries" task card into the screen.
  A faceplate container is created.

3.11.7.7 Generating a HMI user data type as type

Introduction

To define a new HMI user data type, add a new type in the project library.

Procedure

1. Open the "Libraries" task card.
2. Select the "Add new type" command under "Types" in the shortcut menu of the project library.
   A dialog opens.
3. Select "HMI user data type".
4. Specify the device for the new type.
5. Assign a descriptive name to the new HMI user data type.

Result

The new HMI user data type is created and displayed under the selected name in the project library.

The user data type is assigned the status "In progress" and the version 0.0.1.

The editor for the user data type opens.

Using an HMI user data type

The released default version of the type is available in the tag tables for HMI tags as data type for internal tags.

3.11.7.8 Generating HMI user data type from PLC data type

Introduction

To create a new HMI user data type based on a PLC data type, add a PLC data type in the project library.

Requirement

- A PLC is created in the project.
- A PLC data type with at least one element is configured.

Procedure

1. Navigate in the project tree to the PLC and here to the "PLC data types" node.
2. Drag and drop the PLC data type node to the "Types folder" in the project library.
3. In the "Add type" dialog you specify the properties of the new type.

Result

The new HMI user data type is created and displayed under the selected name in the project library.

The User data type is assigned the status "Released" and the version 0.0.1.

Editing HMI user data type based on PLC data type

1. Select "Edit type" in the shortcut menu of the type.
   This can be occur in the project tree or in the "Libraries" task card.
2. To open the editor for editing, you must set a test environment.
   The version is in the "In test" state and can be edited.
3. Release the version after editing.

Using an HMI user data type

- To use the PLC data type stored in the library at another PLC, drag and drop the node from the library to the "PLC data types" node of a PLC in the project tree.

  Note

  PLC data types can only be instantiated at PLCs of the same type.

  The PLC data type is available as data type in the tag tables for PLC tags.

The new HMI user data type can be stored in a global library.

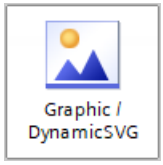
3.11.7.9 Creating a graphic and dynamic SVG as type

Introduction

To define a new graphic type or define a dynamic SVG as a type, add a new type in the project library.

Procedure

1. Open the "Libraries" task card.
2. Select the "Add new type" command under "Types" in the shortcut menu of the project library.
   A dialog opens.
3. Select "Graphic/Dynamic SVG".



4. Select whether you are adding a graphic or a dynamic SVG as type.

Result

The new type is created and displayed in the project library. The type is assigned the status "In progress" and the version 0.0.1.

For graphics, the editor for graphics (Page 238) opens.

For dynamic SVG graphics, the editor for dynamic SVG graphics (Page 182) opens and a dynamic standard SVG is displayed.

Graphics and dynamic SVGs can be instantiated in screens and faceplates.

3.11.7.10 Editing dynamic SVG type

Create dynamic SVGs as type to be able to exchange and update SVGs conveniently.

The type instance concept is implemented for dynamic SVGs. The selection of the dynamic SVG is defined in the type, and the dynamization of the properties takes place at the instance.

A type contains exactly one dynamic SVG.

Dynamic SVGs are language-neutral.

In contrast to using SVGs from the "Graphics" task card, using types in faceplates allows different SVGs with the same name in one faceplate.

Note

Version compatibility of dynamic SVG types

Dynamic SVG types have been introduced with WinCC Unified V18. They cannot be used with faceplates < V18. They cannot be used with device versions or project versions < V16.

If you load a project that uses an instance of a dynamic SVG type in a faceplate into a Runtime V17, this results in an error during loading.

Note

Faceplates and dynamic SVGs

If you create a faceplate type from a group of selected screen objects, dynamic SVGs are not included. To insert a dynamic SVG into a faceplate, open the faceplate type in the editor and insert the SVG from the library here.

Note

Manual assignment of a type version

If you manually assign a type version to a dynamic SVG type or rename the type, a DELTA download is no longer possible.

Introduction

After a new type is created, a dynamic standard SVG is displayed. You replace this SVG with the SVG you need.

No properties are displayed for the standard SVG.

To replace an SVG that has already been created or edit its properties, open the type in the editor.

Procedure

1. To replace the dynamic standard SVG or the SVG of the previous version, select "Replace with other dynamic SVG" in the shortcut menu of the SVG.
   Other shortcut menu options are not available in V18.
2. Select a dynamic SVG in the file system.
   The existing SVG is replaced by the SVG you selected.
   For an instance of the dynamic SVG type, the properties are displayed in the Inspector window of the image editor or faceplate editor. It is possible to dynamize properties here.

If the selected SVG cannot be fully displayed in the visible area of the screen, scroll bars will be displayed so that the view can be shifted horizontally and vertically.

If you select a file that does not contain a valid dynamic SVG, an error message is output in the Inspector window.

Release version

1. To release the version of the type, open the note at the top.

2. When releasing the type, enable the "Set dependent types to edit mode" option.
   Faceplate types in which the dynamic SVG is used are set to edit mode and thus use the current version of the dynamic SVG.

Notes on dynamization

- The dynamization of properties, e.g. color, affects the instance, not the type.
- The dynamization of properties is retentive if the new SVG has the same properties as the previous version when the type is changed.
  If the properties of the new SVG differ from the previous version, default values are used. In this case, no dynamization is specified.
- When a type is instantiated in a screen, the "Contained type - name" property in the Inspector panel under "Properties > Miscellaneous" shows the name and version of the instantiated type.

3.11.7.11 Creating a script module as a type

Introduction

To define a new script module type, add a new type in the project library.

Procedure

1. Open the "Libraries" task card.
2. Select the "Add new type" command under "Types" in the shortcut menu of the project library.
   The "Add new type" dialog opens.
3. Select "Script module".
4. Define the lowest device version.

Result

The new script module type is created and displayed in the project library.

The script module type is assigned the status "In progress" and the version 0.0.1.

The editor for a script module opens.

Working with script modules

In the editor for script modules, you create functions in which you can define parameters. You select these functions when dynamizing screens, screen images or tasks via events in the function list. If you change or delete functions or parameters in the script module type, the referenced functions will be adapted automatically.

To check the syntax, click on  "Syntax check" in the editor.

Error messages that allow you to analyze the script are displayed in the Inspector window under "Info > Compile".

When creating scripts, you are supported by snippets that you access from the shortcut menu under "Snippets" and which contain logic blocks for the following groups:

- Faceplate
- HMIRuntime
- Logic

See also

"Scripts" editor (Page 981)

3.11.8 Using master copies

3.11.8.1 Basics

You can save objects that you have created in the project tree as a master copy in a library. Based on the master copy, you can create a new object in the project tree.

Master copies allow you to efficiently create the same or similar objects. By using master copies in a global library, these objects can be used in other projects.

Basics

You can use all objects as master copy, which you can create new in the project tree:

- HMI devices, PLCs, technology objects, ...
  and the objects contained in these objects, e.g. cams
- Screens
- Tag tables
- Parameter set types
- Scripts

You can use objects as master copy, which you can create new in a work area:

- Tags
- HMI alarms
- Tasks

- Cycles
- Text and graphic lists

Objects can be added as a master copy in the project library or in an open global library.

Basic procedure

Once you have used this method to create and edit an object that you want to use it as a master copy, drag and drop it onto the "Master copy" folder in a library.

Then drag and drop the master copy back into the project tree to create a new object that you can further customize.

3.11.8.2 Using a script as a master copy

Requirement

- A project is open.
- An HMI device has been created and opened.
- The project tree is open.
- The "Libraries" task card is opened.

Procedure

You can store a global module, a global definition area or a function as a master copy.

1. Open the "Scripts" editor in the project tree.
2. Drag and drop a script into the "Master copies" folder of a library.

Result

You have created a master copy from a script in the library.

Using a master copy

1. Drag and drop the master copy from the library into the "Scripts" folder in the project tree. A new script is created.
2. Edit the script.

3.11.8.3 Using a screen as a master copy

Requirement

- A project is open.
- An HMI device has been created and opened.

- The project tree is open.
- The "Libraries" task card is opened.

Procedure

1. Open the "Screens" editor in the project tree.
2. Drag and drop the screen into the "Master copies" folder of a library.

Result

In a library you have created a master copy from a screen.

Using a master copy

1. Drag and drop the master copy from the library into the "Screens" folder in the project tree. A new screen has been created.
2. Edit the screen.

3.12 Using WinCC version compatibility

3.12.1 Basics on version compatibility

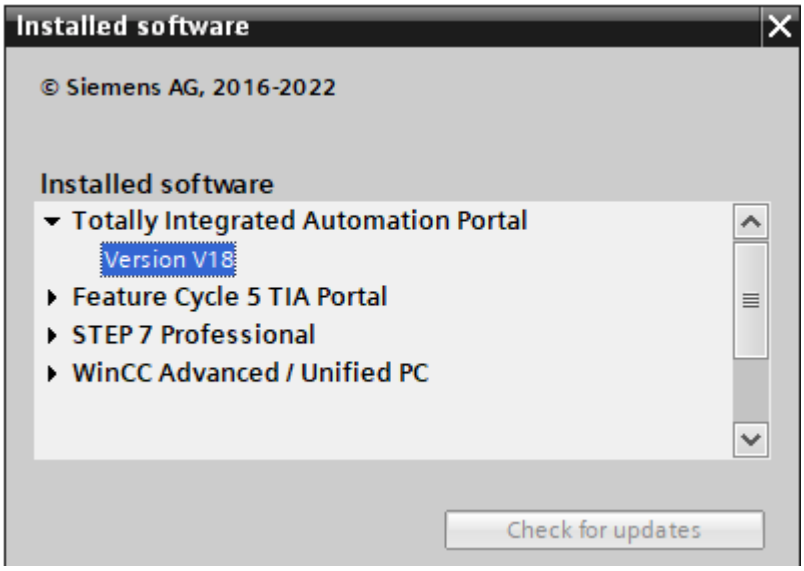
Introduction

The interaction of the following versions is of importance for version compatibility:

- WinCC version
- Project version
- Configured Runtime version
- Installed Runtime version

WinCC version

The WinCC version is the WinCC version installed on the configuration PC for TIA Portal, for example, WinCC Unified V18.

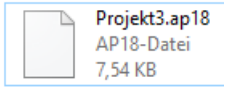


The installed version is displayed under "Help > Installed software ...".

Project version

The project version is the version of a WinCC project.

- When you create a new WinCC project, the project version is always the same as the WinCC version.
- A project successfully opened in WinCC Unified always has the version of the software used. Projects with older versions are automatically upgraded at the time of opening.
- An existing project can have a project version that is older than the WinCC version. You can make out the version of a project from the file ending of the project file.



Note

To open a WinCC project that was created with an older version of TIA Portal, you must first upgrade its project version to the WinCC version of the currently installed TIA Portal.

After the upgrade, the functions of the current WinCC version are available in the project. You can then open, edit, save, compile, download or simulate the project.

You can find additional information at AUTOHOTSPOT.

If the project version cannot be changed, open the project in a TIA Portal whose WinCC version is the same as the project version. There, you edit, save, compile, download or simulate the project.

Configured Runtime version

The configured Runtime version is the version of the runtime configured in a WinCC project for an HMI device.

A WinCC project can contain HMI devices with differently configured Runtime versions. HMI devices whose configured Runtime version is older than the current version can be restricted in their functions when compared to the HMI devices with the current version.

The configured Runtime version must be compatible with the Runtime version installed on the target device for the project to be downloaded.

When you add a new HMI device to a WinCC project, its pre-selected, configured Runtime version is always the highest available Runtime version.

To add a new device with an older version, select the desired version in the "Add new device" dialog.

You can change the configured Runtime version in the "Devices & Networks" editor or the device properties in the project tree.

You can find additional information at Changing the configured runtime version (Page 206).

Installed Runtime version

The installed Runtime version is the version of Runtime installed on the HMI device. The version is displayed on the HMI device. You can find information on this in the documentation of the hardware.

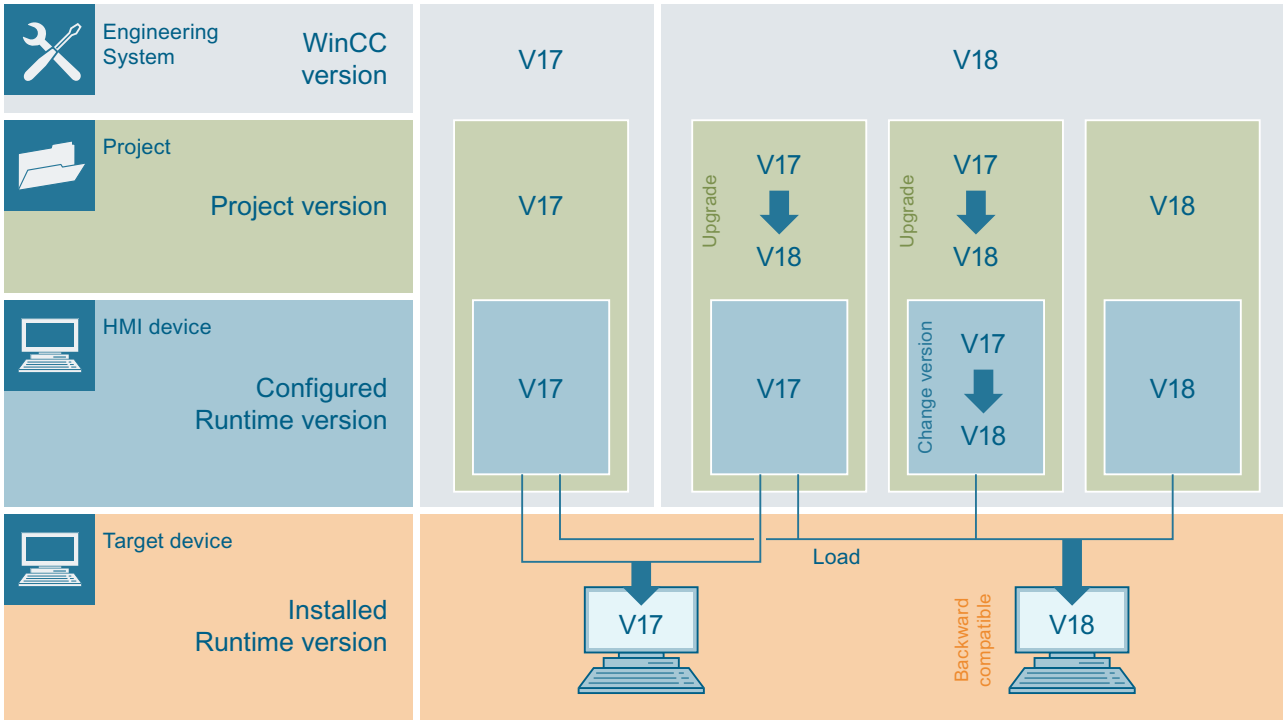
An installed Runtime version V18 or higher supports backward compatibility up to V17.

You can find more information about the installed Runtime version under:

- Installed Runtime version for Unified Comfort Panel (Page 191)
- Installed Runtime version for Unified PC (Page 192)

Version compatibility

The following diagram shows the interaction of the different versions:



When downloading to a target device, make sure that the configured and the installed Runtime versions are compatible.

TIA Portal checks the compatibility during the download. The following applies when incompatible versions are detected:

- Unified Comfort Panel: In the "Load preview" dialog, you are given the option to install an image with a compatible version on the Panel.
  You can find additional information at Basics for downloading projects (Page 7147).
- Unified PC: Download is not possible

Note

When loading from an external storage medium, compatibility is checked while you load the project to the device (Unified PC).

Simulation

With an installed Runtime as of version V17, backward compatibility is also supported for simulations.

You can simulate Runtime projects with a configured Runtime version of V16 onwards.

3.12.1.1 Installed Runtime version for Unified Comfort Panel

Note

Changing the installed Runtime version deletes all data on the Unified Comfort Panel

When an image is installed, the data on the Unified Comfort Panel is deleted.

Back up the following data before you install an image.

- Database for the tag persistence
- Data logs
- Alarm logs
- Parameter sets

For Unified Comfort Panels, the operating system and the runtime are bundled into one image with a device version that can be transferred to the HMI device if necessary. The device version determines which version of the operating system and Runtime can be installed with the image.

The Runtime version installed with the image on the Unified Comfort Panel must be compatible with the configured Runtime version for the project to be downloaded.

Note

Installing image with compatible version

There are several ways to install an image with a compatible version:

- When TIA Portal detects an incompatible runtime version during online loading, you have the option in the "Load Preview" dialog to install an image with a compatible, installed Runtime version on the Unified Comfort Panel before the download.
  You can find additional information at Basics for downloading projects (Page 7147).
- Independent of the download, you have the following options for installing an image:
  - Updating the operating system of the HMI device (Page 7167)
  - Updating the operating system of the HMI device from a data storage medium (Page 7169)

Backwards compatibility

HMI devices starting from the installed Runtime version V17 are backward compatible. Runtime projects with a configured Runtime version V16 or higher can run on them.

Note

Conversion of the logging data with backward compatibility

When a project is started for the first time in runtime in the backward compatibility mode, its logging data is adapted to the new database schema. This process is irreversible.

Example:
A project with configured Runtime V17 is loaded to a device on which Runtime V18 is installed. During the project start, the logging data is adapted to match the database schema of V18. You can no longer transfer the log to a device with Runtime V17 and run it there.

See also

Basics on version compatibility (Page 187)

Basics for downloading projects (Page 7147)

Updating the operating system of the HMI device (Page 7167)

Updating the operating system of the HMI device from a data storage medium (Page 7169)

3.12.1.2 Installed Runtime version for Unified PC

For Unified PC, the operating system and Runtime are installed independently of each other. The installed Runtime version must be compatible with the configured Runtime version.

Backwards compatibility

HMI devices starting from the installed Runtime version V17 are backward compatible. Runtime projects with a configured Runtime version V16 or higher can run on them.

Note

Conversion of the logging data with backward compatibility

When a project is started for the first time in runtime in the backward compatibility mode, its logging data is adapted to the new database schema. This process is irreversible.

Example:

A project with configured Runtime V17 is loaded to a device on which Runtime V18 is installed. During the project start, the logging data is adapted to match the database schema of V18. You can no longer transfer the log to a device with Runtime V17 and run it there.

See also

Basics on version compatibility (Page 187)

3.12.1.3 Use cases

Introduction

When commissioning a plant, the following versions are usually identical:

- WinCC version
- Project version
- Configured Runtime version
- Installed Runtime version

If you want to replace an HMI device or expand your system, the versions can differ. In such cases, WinCC version compatibility helps you to continue operating your plant with as little modification work as possible.

Example: Replacing the HMI device

Your system contains a Unified Comfort Panel with an installed Runtime version V17.

You want to replace this Unified Comfort Panel with another Unified Comfort Panel with an already preinstalled Runtime version V18. Nothing should be changed in the configuration.

To be able to download your Runtime project V18 to the HMI device, use the backwards compatibility.

More information can be found in the section Replacing a Unified Comfort Panel (Page 210).

Example: Expanding an existing system with an HMI device with an installed Runtime version

You want to expand your system with a Unified Comfort Panel with an installed Runtime version V17.

You are already using WinCC V18 on your configuration PC.

To use the HMI device, follow these steps:

1. Upgrade the project version.
2. Configure a Unified Comfort Panel V17 in the Engineering System.
3. Compile and download the Runtime project to the Unified Comfort Panel with an installed Runtime version V17.

Example: Expanding an existing system with an HMI device with a later Runtime version installed

You want to expand your system with a Unified Comfort Panel with an installed Runtime version V18 and use the functionalities of the current version.

You are using WinCC V17 on your configuration PC.

To use the HMI device, follow these steps:

1. Install WinCC V18 on your configuration PC.
2. Upgrade the project version.

3. Configure a Unified Comfort Panel V18 in the Engineering System.
4. Compile and download the Runtime project to the Unified Comfort Panel with an installed Runtime version V18.

Example: Upgrading an HMI device

Your system contains a Unified PC with an installed Runtime version V16 or V17.

You want to use the functionalities of the current version on the Unified PC.

You can find additional information at Upgrading a Unified PC (Page 207).

See also

Replacing a Unified Comfort Panel (Page 210)

Upgrading a Unified PC (Page 207)

Upgrading a Unified Comfort Panel (Page 208)

Replacing a Unified PC (Page 212)

3.12.2 Upgrade project

Introduction

To open a WinCC project that was created with an older version of the Engineering System, you must first upgrade the project version to the WinCC version of the currently opened Engineering System.

Requirements

- The project version is a predecessor of your WinCC version.

  Note

  No version skips

  You cannot skip any version when upgrading. A project with version V16 must first be upgraded in a TIA Portal version V17 before you can upgrade it in a TIA Portal version V18 to the current version.

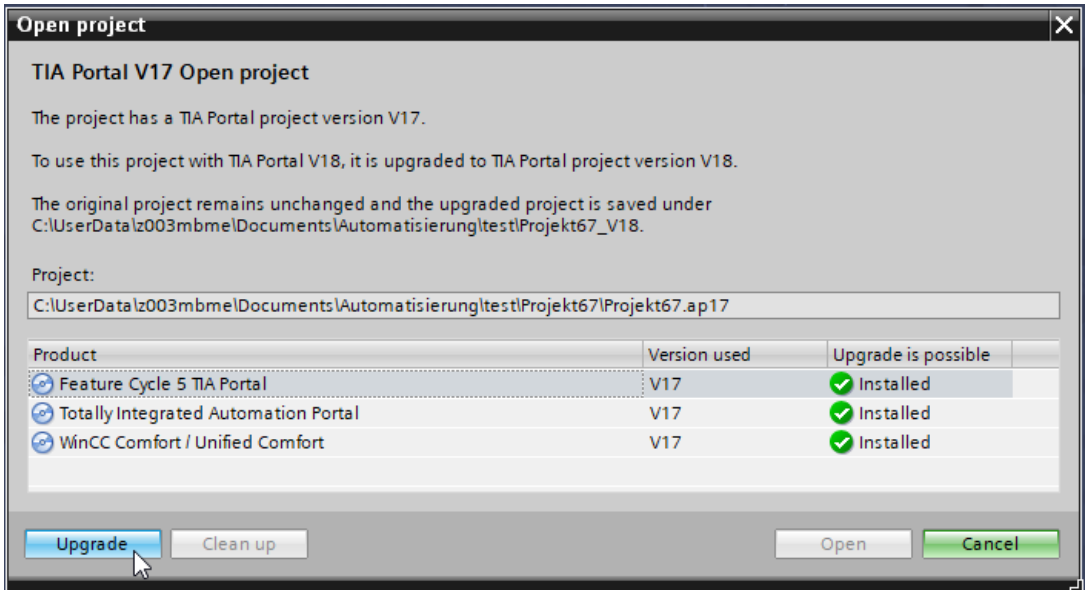
- You have write access to your project drive.
- The project drive has sufficient storage capacity for another project of this size.

Procedure

To upgrade a project from TIA Portal V17 or higher to your WinCC version, follow these steps:

1. Select the "Open" command in the "Project" menu.
   The "Open Project" dialog box opens and the list of recently used projects is displayed.
2. Select a project from the list and click "Open".
3. If the project is not included in the list, click the "Browse" button. Navigate to the project folder and open the project file.
   The "Open Project" dialog opens.

4. Click "Upgrade".



The project is upgraded to the current project version and opened. The procedure can take time, depending on the number of devices and objects used in the project.

Note

Unsupported devices

If there are devices in the project that are not supported by the current version, an upgrade is not possible.

Open the project in the version of the TIA Portal in which it was created, and remove the unsupported devices or replace them with supported devices.

More information: Unsupported devices (Page 198)

Note

Devices not fully supported

If devices are configured in the project that are only partially supported, an upgrade is possible.

In the upgraded project, replace the devices with fully supported devices.

More information: Devices not fully supported (Page 197)

5. Check whether devices and library objects have to be adapted to the new version.

Result

- The content of the old WinCC project is saved in a new project with the current project version.
  The original project is not overwritten and can still be edited with a compatible older version of the TIA Portal.
- You can open, edit, save, compile, download or simulate the project.

- All functions of the current WinCC version are available in the project.
- If necessary, match the Runtime version of the HMI devices.

See also

Upgrading the installed Runtime version of a device (Page 207)

3.12.3 Devices not fully supported

Error during upgrade

If a project contains devices or device versions that are no longer fully supported in the current version of WinCC, the project can be upgraded to the current version of WinCC.

Unsupported devices may be present in a project in the following locations:

- Devices in the project tree
- Devices in the master copies of the project library

Devices that are no longer fully supported are flagged with a symbol: 

If you compile a device that is no longer complete, you get an error message:

- The device is not supported. Compilation is therefore not possible. Please switch to a supported device.
- The Runtime version is not supported. Compilation is therefore not possible. Switch to a supported version.

Changing the device type or the device version in the project tree

To change the device type or the device version of a device in the project tree, follow these steps:

1. Select the "Change device/version" command in the shortcut menu of the outdated device.
2. In the "Change device" dialog, select as new device a device or device version that is supported in the current version of WinCC.

Devices and device versions not fully supported

KTP400 Basic 2nd Generation	15.0.0.0
KTP700 Basic 2nd Generation	15.0.0.0
KTP900 Basic 2nd Generation	15.0.0.0
KTP1200 Basic 2nd Generation	15.0.0.0
KTP400F Mobile	15.0.0.0
KTP700 Mobile	15.0.0.0
KTP700F Mobile	15.0.0.0
KTP900 Mobile	15.0.0.0
KTP900F Mobile	15.0.0.0

Mobile Panel 177	
Mobile Panel 277	
KP400 Comfort	15.0.0.0
KP700 Comfort	15.0.0.0
KP900 Comfort	15.0.0.0
KP1200 Comfort	15.0.0.0
KP1500 Comfort	15.0.0.0
KP1500 Comfort V2	15.0.0.0
KTP400 Comfort	15.0.0.0
TP700 Comfort	15.0.0.0
TP700 Comfort INOX PCT	
TP700 Comfort Outdoor	15.0.0.0
TP900 Comfort	15.0.0.0
TP900 Comfort INOX PCT	15.0.0.0
TP1200 Comfort	15.0.0.0
TP1200 Comfort INOX PCT	15.0.0.0
TP1200 Comfort PRO	15.0.0.0
TP1500 Comfort	15.0.0.0
TP1500 Comfort Outdoor	15.0.0.0
TP1500 Comfort V2	15.0.0.0
TP1900 Comfort PRO	15.0.0.0
TP2200 Comfort	15.0.0.0
TP2200 Comfort V2	15.0.0.0

You can find detailed compatibility lists of all HMI devices using the compatibility tool (https://support.industry.siemens.com/cs/ww/en/view/64847781).

3.12.4 Unsupported devices

Error during upgrade

If a project contains devices or device versions that are no longer supported in the current version of WinCC, upgrading is not possible. To upgrade the project to the current version, you need to open the project in an older version of WinCC and change the device type or version to a value supported by the current version of WinCC.

Devices that are no longer supported are flagged with a symbol: 

Unsupported devices may be present in a project in the following locations:

- Devices in the project tree
- Devices in the master copies of the project library

Changing the device type or the device version in the project tree

To change the device type or the device version of a device in the project tree, follow these steps:

1. Select the "Change device/version" command in the shortcut menu of the outdated device.
2. In the "Change device" dialog, select as new device a device or device version that is supported in the current version.

Changing the device type or device version in the library

To change the device type or the device version of a device in the project library, follow these steps:

1. Search for the outdated device in the master copies of the project library.
2. Copy the master copy from the project library to the project tree.
3. Make a note of the name of the master copy. Delete the master copy in the project library.
4. Select the "Change device/version" command in the shortcut menu of the outdated device.
5. In the "Change device" dialog, select as new device a device or device version that is supported in the current version of WinCC.
6. Create a new master copy of the project library from this device.
7. Give the new master copy the same name as the deleted master copy.

Devices and device versions not supported

KTP400 Basic 2nd Generation	13.0.0.0, 14.0.0.0
KTP700 Basic 2nd Generation	13.0.0.0, 14.0.0.0
KTP900 Basic 2nd Generation	13.0.0.0, 14.0.0.0
KTP1200 Basic 2nd Generation	13.0.0.0, 14.0.0.0
KTP400F Mobile	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KTP700 Mobile	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KTP700F Mobile	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KTP900 Mobile	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KTP900F Mobile	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
Mobile Panel 177	11.0.0.0, 12.0.0.0
Mobile Panel 277	11.0.0.0, 12.0.0.0
KP400 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KP700 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KP900 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KP1200 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
KP1500 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 13.0.1.0, 14.0.0.0
KP1500 Comfort V2	11.0.0.0, 12.0.0.0
KTP400 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
TP700 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
TP700 Comfort INOX PCT	11.0.0.0, 12.0.0.0
TP700 Comfort Outdoor	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0

TP900 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
TP900 Comfort INOX PCT	11.0.0.0, 12.0.0.0, 14.0.0.0
TP1200 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 14.0.0.0
TP1200 Comfort INOX PCT	11.0.0.0, 12.0.0.0, 14.0.0.0
TP1200 Comfort PRO	11.0.0.0, 12.0.0.0, 14.0.0.0
TP1500 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 13.0.1.0, 14.0.0.0
TP1500 Comfort Outdoor	11.0.0.0, 12.0.0.0, 14.0.0.0
TP1500 Comfort V2	11.0.0.0, 12.0.0.0
TP1900 Comfort PRO	11.0.0.0, 12.0.0.0
TP2200 Comfort	11.0.0.0, 12.0.0.0, 13.0.0.0, 13.0.1.0, 14.0.0.0
TP2200 Comfort V2	11.0.0.0, 12.0.0.0
WinCC Runtime Advanced	11.0.0.0, 12.0.0.0

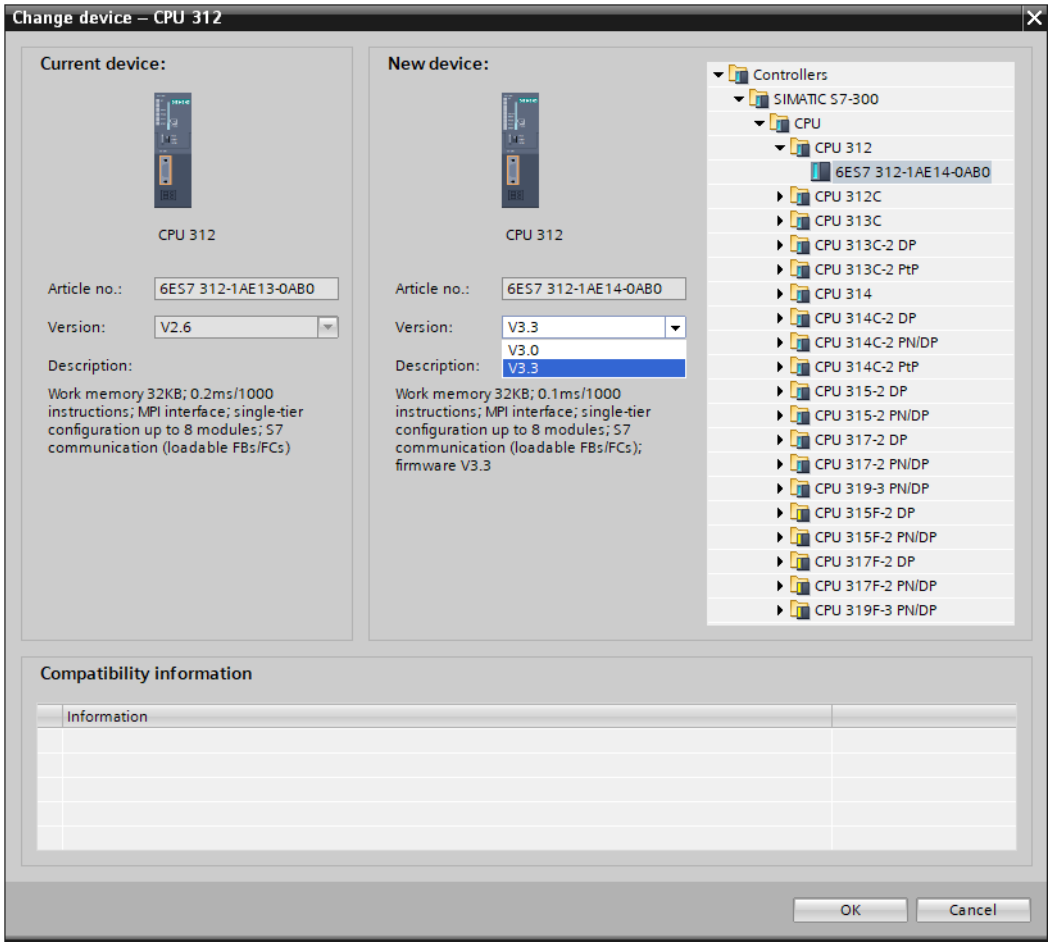
You can find detailed compatibility lists of all HMI devices using the compatibility tool (https://support.industry.siemens.com/cs/ww/en/view/64847781).

3.12.5 Matching objects after upgrading

After the successful upgrading of a project, you must match the versions of individual devices and objects if required.

Matching devices after upgrading

1. In the project tree for a device, in the shortcut menu, select "Replace device ....".
2. Check whether the device is available in a newer version.



More information:

- Replacing the configured HMI device (Page 202)
- Replacing a device (Page 209)

3. Then compile each device in the project.

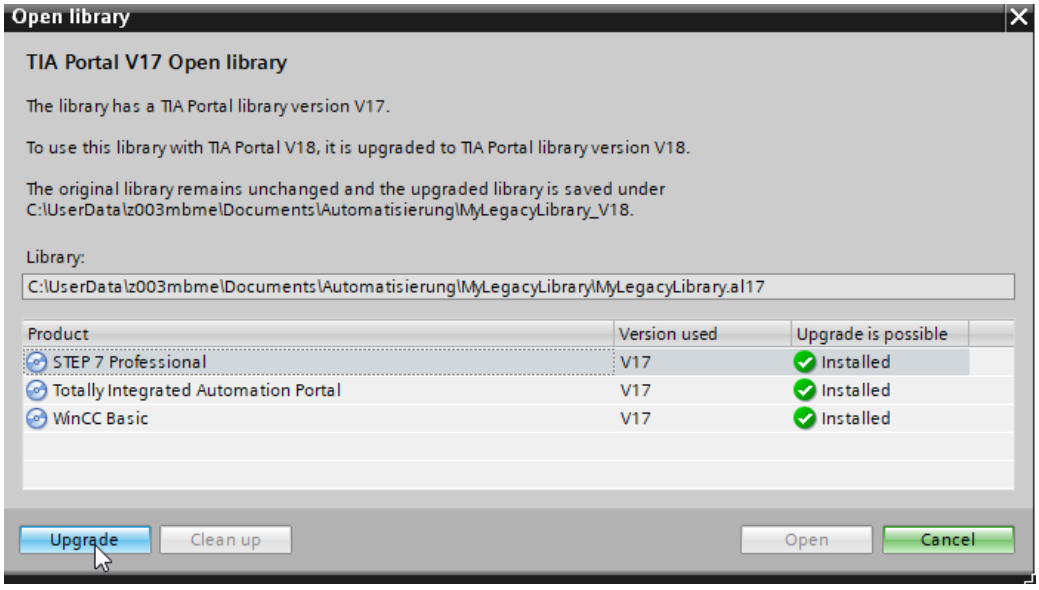
Note

Sequence of compilation

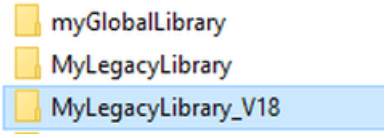
Compile the PLCs first and then the HMI devices in your project. In this way, you ensure that the controller data required to compile the HMI devices is available.

Upgrading global libraries

1. Open a global library that was created with an older version of the TIA Portal.
   The "Open Library" dialog is displayed.



2. Click "Upgrade".
   The library is upgraded and saved as a new global library with the appendage "_V18".
   The library of the predecessor version is retained.



See also

Upgrading a global library (Page 205)

3.12.6 Replacing the configured HMI device

3.12.6.1 Basics for replacing the configured HMI device

Introduction

When you replace the devices, you can use existing configurations for your new HMI devices.

For example, you replace a Unified PC for a Unified Comfort Panel or a 7" Unified Comfort Panel for a 12" Unified Comfort Panel.

All data configured by you is retained in the configuration data. This means you do not need to copy individual objects of one device and paste them to another.

Note

When you replace a Unified Comfort Panel and select a PC station as your new device, for example, WinCC Unified PC RT is automatically moved below the PC Station in the project tree.

Adjusting the screen size

If the new device supports a different resolution than the previous device when you replace a device, adjust the screen size.

Customizing a connection

When you have configured a connection to a PLC and are replacing the HMI device with a different device type, error messages may occur, for example "The object 'IE general' is not supported in the new configuration and will be removed".

This may happen, for example, when you replace a Unified PC with a Unified Comfort Panel because the configured communications module is not supported after the exchange.

Therefore, check the configured connections and make adaptations if necessary.

See also

Basics on version compatibility (Page 187)

3.12.6.2 Replacing the configured HMI device

Requirement

- A project has been created and opened.
- A Unified PC or a Unified Comfort Panel is used in the project.
- The screens have been adapted.

Procedure

To replace an HMI device with another HMI device, follow these steps:

1. In the project tree, select the HMI device to be replaced.
   Alternatively, select the HMI device to be exchanged in the "Devices & Networks" editor.
2. Depending on the device, select "Change device / version" or "Change device" from the shortcut menu.
   The "Change device" dialog opens.

3. Select the desired device.
   Details of hardware differences can be found in the "Compatibility information".
   If necessary, adapt the version.
   You can find additional information at Changing the configured runtime version (Page 206).
4. Confirm the dialog.

Result

You have replaced the HMI device used in the project.

If needed, make the following adaptations:

- If you have exchanged a Unified Comfort Panel with a Unified PC or vice versa, for example, you need to adapt the configured connections.
- If you have selected an HMI device with a different screen resolution, adjust the screens.

Further procedure

To complete the device change, the following further steps are necessary:

- If necessary, replace the HMI device in your plant.
- Transfer the stored data to the new HMI device.

  Note

  Data transfer

  Note that when changing the device type (changing from Unified Comfort Panel to a Unified PC or vice versa), the transfer of databases is not supported.

  Data of the central user administration (UMC) cannot be transferred and must be loaded together with the Runtime project.

- Compile and load the Runtime project into the new HMI device.

See also

Changing the configured runtime version (Page 206)

3.12.6.3 Adapting the configuration of the connection

Introduction

If an HMI device is changed, error messages may occur, for example "The object 'IE general' is not supported in the new configuration and will be removed".

These alarms refer to configured connections of the device and are triggered, for example, by different interfaces when replacing the Unified Comfort Panel with a Unified PC.

These connections are marked red after a device replacement. If you would like to continue to use these connections, you have to adapt the configuration of the connection.

Procedure

To connect a PLC to the HMI device again after the device replacement, follow these steps:

1. Open the "Devices and Networks" editor.
2. Click "Network" in the toolbar of the network view.
3. If necessary, add a communications module.
4. Network the interface of the HMI device with the interface of the PLC.
5. In the table area of the network view, click on the "Connections" table.
6. Select the connection marked red.
7. Enter the new interface under "Properties > General > General > Connection path" in the Inspector window.

3.12.7 Upgrading a global library

Introduction

To process objects of a global library in a project, you must first upgrade the version of the global library to the project version. You are prompted accordingly when you open the global library.

Requirements

- The version of the global library is a predecessor version of your project version.
- You have write access to your project drive.
- All types in the library have been released.

Procedure

To upgrade a global library from TIA Portal V16 or V17, proceed as follows:

1. Open the global library.
   The "Upgrade global library" dialog box opens.
2. Click "OK".

Result

A copy of the global library is created and upgraded. The global library opens.

3.12.8 Changing the configured runtime version

Introduction

If you want to upgrade or exchange the HMI device, change the configured Runtime version of an HMI device.

Change, for example, the configured Runtime version of a Unified Comfort Panel from V17 to V18.

Requirements

- A project has been created and opened.
- The project contains an HMI device.

Procedure

To change the configured Runtime version, follow these steps:

1. Double-click on "Devices & Networks" in the project tree.
   The editor opens.
2. Select the desired HMI device in the device view.
3. Select "Change device/version" in the device shortcut menu of the HMI device.
   A dialog opens.
4. Select the required HMI device.
5. Depending on the Runtime version installed on the target device, select a compatible Runtime version under "Version".

   Note

   Selection of Runtime versions

   The project version determines which Runtime versions are offered to you.

6. Confirm your selection with "OK".

Alternatively, you can also change the configured Runtime version in the device properties in the project tree.

Result

You have changed the configured Runtime version of the HMI device in the WinCC project.

To successfully download the project, a compatible Runtime version must be installed on the target device.

In the next step, you upgrade the installed Runtime version of the HMI device or exchange the device.

See also

Replacing the configured HMI device (Page 203)

Replacing the configured HMI device (Page 202)

3.12.9 Upgrading the installed Runtime version of a device

3.12.9.1 Upgrading a Unified PC

Introduction

To use the new functionalities of the current Runtime version, you must upgrade the Unified PC.

Requirement

- No project is running in Runtime on the Unified PC.

Note

No version skips

You cannot skip any version when upgrading.

Procedure

1. Install the higher Runtime version on the Unified PC.
   The existing data is retained, e.g. logs and parameter sets.

   Note

   Converting logging databases

   When a project is started for the first time in Runtime after upgrading, its logging data is adapted to the new database schema. This process is irreversible.

2. Upgrade the engineering system to the WinCC version that corresponds to the Runtime version installed on the Unified PC.
3. Upgrade the project version of the project to the WinCC version.
4. Upgrade the configured Runtime version of the HMI device.
5. Compile and download the project into the Unified PC.
6. Start the project on the Unified PC in Runtime.

See also

Use cases (Page 193)

Changing the configured runtime version (Page 206)

3.12.9.2 Upgrading a Unified Comfort Panel

Introduction

To use the new functionalities of the current Runtime version, you must upgrade the Unified Comfort Panel.

Note

Protection against loss of data

To simplify data transmission, save the following data to external storage media:

- Database for the tag persistence
- Data logs
- Alarm logs
- Parameter sets

Data from central user administration (UMC) cannot be backed up and must be loaded with the Runtime project.

Note

No version skips

You cannot skip any version when upgrading.

Requirement

- No project is running in Runtime on the Unified Comfort Panel.

Procedure

To upgrade a Unified Comfort Panel, follow these steps:

1. Store the above-mentioned data of the Runtime project on external storage media.
2. Load an image with the higher Runtime version.

NOTICE
Data loss All data stored internally on the Unified Comfort Panel will be lost.

3. Make the backed up data available to the Unified Comfort Panel again.
   Do not save the tag persistence database and the parameter sets on the same storage medium as the logs.

4. Upgrade the engineering system to the WinCC version that corresponds to the Runtime version installed on the Unified Comfort Panel.
5. Upgrade the project version of the project to the WinCC version.
6. Upgrade the configured Runtime version of the device.
7. Compile and load the project into the Unified Comfort Panel.
8. Start the project on the Unified Comfort Panel in Runtime.

See also

Use cases (Page 193)

Changing the configured runtime version (Page 206)

3.12.10 Replacing a device

3.12.10.1 Basics

Introduction

When replacing the Unified HMI devices, you can use existing configurations for your new devices and optimize these configurations with very little manual effort. The configuration data are retained.

If you replace an HMI device, for example, a Unified Comfort Panel, and select a PC station as the new device, the configured data is automatically moved under the PC station. The adaptation of the screens to the new screen size takes place in one step after the device replacement. You can find additional information at "Adjusting screens to the new HMI device (Page 213)".

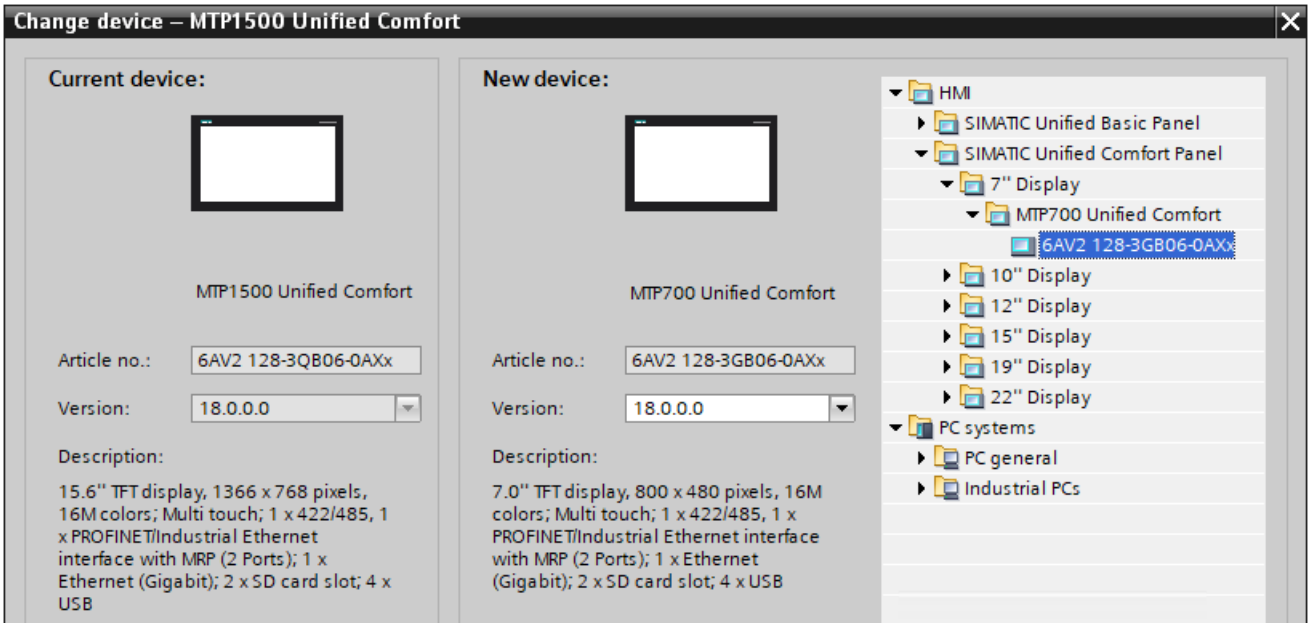
Requirement

- A project has been created and opened.
- In the project, Unified Control Panels or WinCC Unified PCs are used for the visualization.

Introduction

1. Double-click "Devices" in the project navigation.
   The editor opens.
2. Click the required HMI device.

3. Select "Change device/version" in the device shortcut menu.
   A dialog opens.



4. Select the new HMI device and the FW version. Details of hardware differences can be found in the "Compatibility information" at the bottom of the table.
5. Click "OK".
   Replacement of the device is started.

Result

You have replaced the device used in the project.

3.12.10.2 Replacing a Unified Comfort Panel

Introduction

If you replace a Unified Comfort Panel with Runtime version V17 installed, you can replace the device with a Unified Comfort Panel with Runtime version V17 or V18 installed.

HMI devices starting from the installed Runtime version V17 are backward compatible. Runtime projects with a configured Runtime version V16 or higher can run on them. Compile and download existing Runtime projects without any additional adaptation.

Note

Data transfer

Save the following data on external storage media:

- Database for the tag persistence
- Data logs
- Alarm logs
- Parameter sets

Data from central user management (UMC) cannot be backed up and must be loaded with the Runtime project.

Alternatively, replace the Unified Comfort Panel with a Unified PC. To do this, follow the steps under Replacing a Unified PC (Page 212). Note that when replacing the device type, the transfer of databases is not supported.

Requirement

- The desired WinCC version is used.
- The project is available in the engineering system.
- The project version and the WinCC version are identical.

Procedure

Note

Conversion of the logging databases with backward compatibility

When a project is started for the first time in Runtime in the backward compatibility mode, its logging data is adapted to the new database schema. This process is irreversible.

1. Store the above-mentioned data of the Runtime project on external storage media.
2. Make the backed up data available to the new Unified Comfort Panel.
   Do not save the tag persistence database and the parameter sets on the same storage medium as the logs.
3. Load the project in the new Unified Comfort Panel.
4. Start the project on the new Unified Comfort Panel in Runtime.

If you have replaced a Unified Comfort Panel with an installed Runtime version V17 with a Unified Comfort Panel with an installed Runtime version V18, the projects are automatically started in the backward compatibility mode at the start.

Result

You have replaced a Unified Comfort Panel.

The project runs on the new device in runtime.

If needed, make further adaptations, e.g. Upgrading a Unified Comfort Panel (Page 208).

See also

Use cases (Page 193)

3.12.10.3 Replacing a Unified PC

Introduction

If you replace a Unified PC with an installed Runtime version V17, you can replace the device with a Unified PC with an installed Runtime version V17 or V18.

HMI devices starting from the installed Runtime version V17 are backward compatible. Runtime projects whose configured Runtime version is ≥ V16 can run on them. Compile and download existing Runtime projects without any additional adaptation.

Note

Data transfer

Do not save databases directly on a network drive. Power supply can be interrupted at any time. Reliable operation is therefore not guaranteed.

Save the following data on external storage media:

- Database for the tag persistence
- Data logs
- Alarm logs
- Parameter sets
  On Unified PC, parameter sets are saved in the directory of the Runtime project.

Data from central user management (UMC) cannot be backed up and must be loaded with the Runtime project.

Alternatively, replace the Unified PC with a Unified Comfort Panel. To do this, follow the steps under Replacing a Unified Comfort Panel (Page 210). Note that when replacing the device type, the transfer of databases is not supported.

Requirement

- The desired WinCC version is used.
- The project is available in the engineering system.
- The project version and the WinCC version are identical.

Procedure

Note

Conversion of the logging databases with backward compatibility

When a project is started for the first time in Runtime in the backward compatibility mode, its logging data is adapted to the new database schema. This process is irreversible.

1. Install the desired Runtime version on the new PC.
2. Store the above-mentioned data of the Runtime project on external storage media.
3. Make the backed up data available to the new Unified PC.
   On the new PC, select the log directory configured in the "WinCC Unified Configuration" tool during Runtime installation as the storage location for the logs.
4. Compile and download the project into the Unified PC.
5. Start the project on the new unified PC in Runtime.

If you have replaced a Unified PC with an installed Runtime version V17 with a Unified PC with an installed Runtime version V18, the projects are automatically started in backward compatibility mode at the start.

Result

You have replaced a Unified PC.

The project runs on the new device in runtime.

If needed, make further adaptations, e.g. Upgrading a Unified PC (Page 207).

See also

Upgrading a Unified PC (Page 207)

Use cases (Page 193)

3.12.10.4 Adjusting screens to the new HMI device

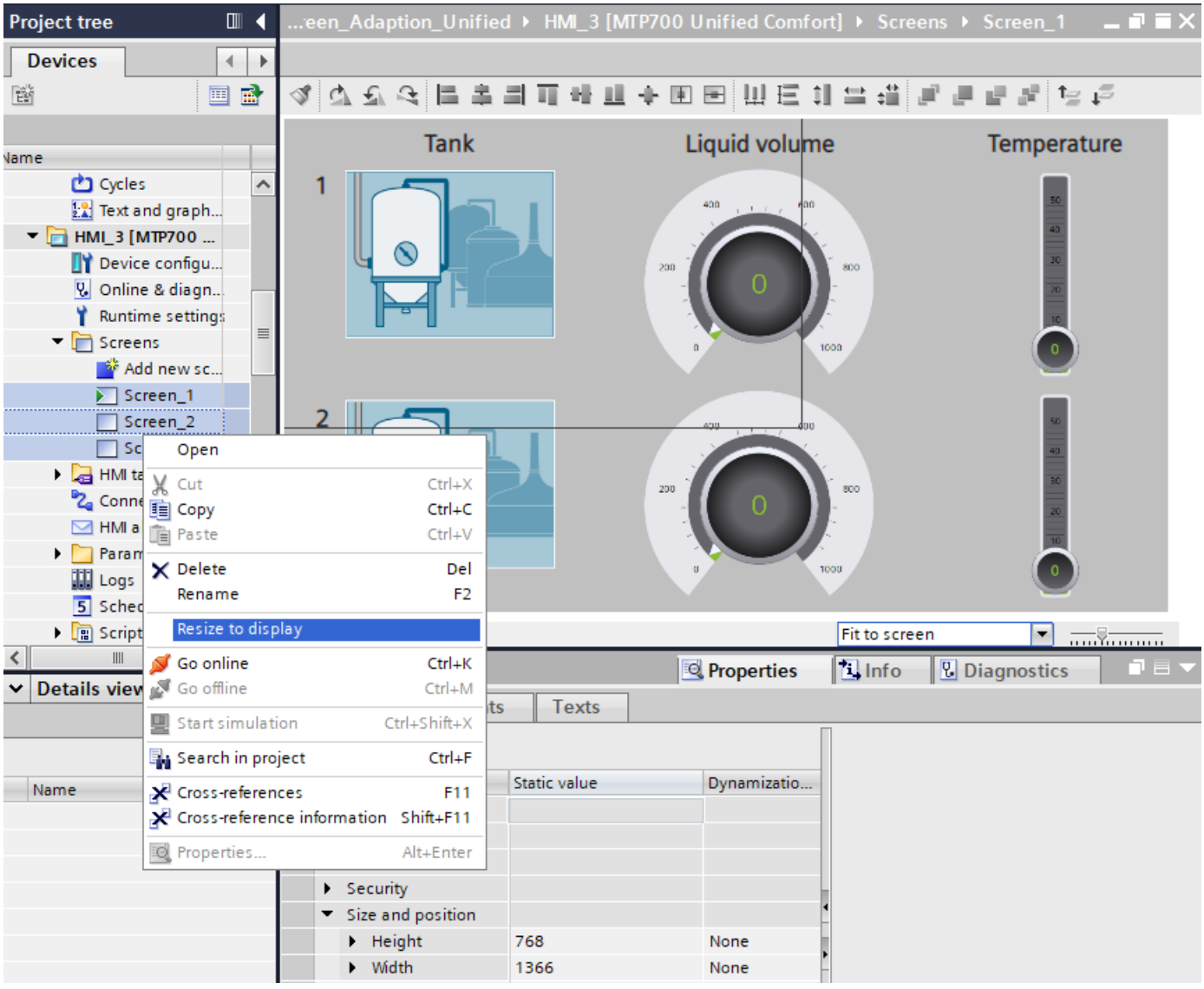
If you replace an HMI device with a smaller or larger display, the configured screen sizes do not fit the new HMI device. The Unified Comfort Panels and Unified PCs offer an easy adaptation of the screen size, the objects and fonts used, all in one step.

Requirement

- A project has been created and opened.
- Unified Control Panels or WinCC Unified PCs are used in the project for the visualization.
- A screen is configured.
- You have exchanged an HMI device for another Unified Control Panel or a WinCC Unified PC.

Adjusting the screen size

1. Double-click "Devices" in the project navigation.
   The editor opens.
2. Click the HMI device.
3. Select a screen under "Screens" and, on the bottom edge of the window, set the scale "Fit to screen".
   - If the new screen size is smaller than the screen size of the exchanged device, a part of the screen is framed by a thin black line. In this case, only the part in the frame would be shown on the display of the new device.
   - If the new screen size is larger than the screen size of the exchanged device, the screen only fills a part of the display area.

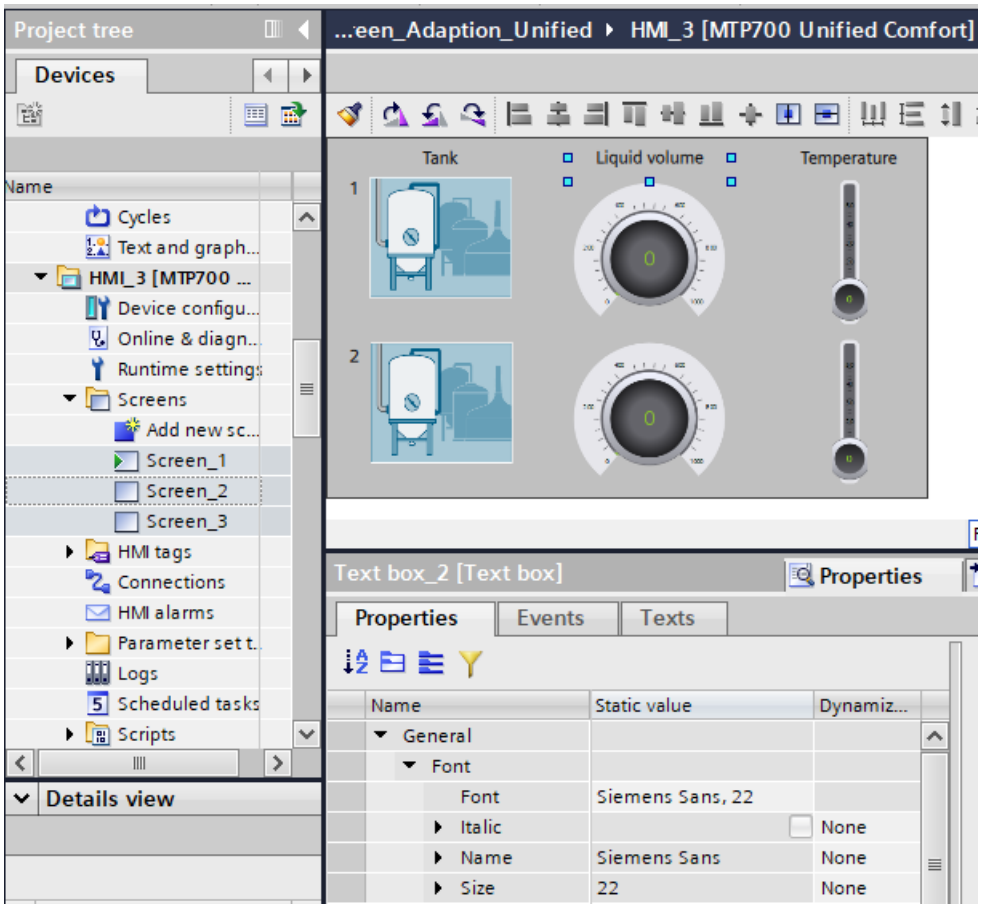


4. Select a screen, a screen group or several screens. Right-click the selected screens and select "Resize to display".

If you scale a large number of screens, an information window is shown that informs you about the progress of the change.

Result

The screen size, the objects, and fonts used are scaled in one step in all the selected screens.



Note

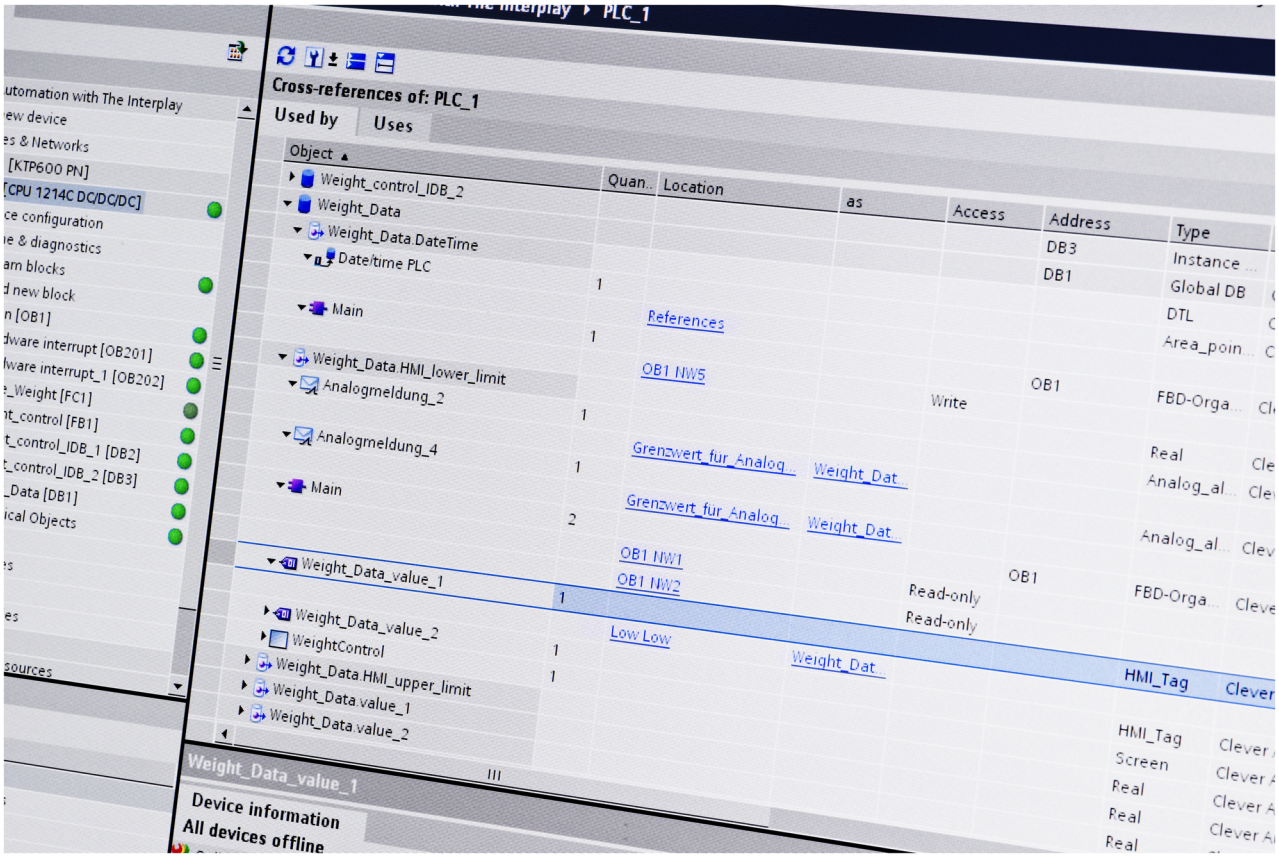
After replacing the HMI device with a substantially larger or smaller variant, check the appearance of the configured screens. Changed display sizes can result, for example, in the fonts being too small or too large.

3.13 Using cross-references

3.13.1 General notes about cross-references

Introduction

The cross-references provide an overview of the use of objects and devices within the project and the project library. The relationships and the dependencies between the objects can be displayed using the cross-references.



- To open the "Cross-references" editor, select "Cross-references" in the shortcut menu of the navigation object in the project tree.
  - or -
  Press <F11>
  - or -
  In the Inspector window, select "About > Cross-references".
- To open the "Cross-reference information", select "Cross-reference information" in the shortcut menu of an object, tag, alarm, etc.
  - or -
  Press <⇑ + F11>.

Benefits of cross-references

Cross-references offer the following advantages:

- When creating the project and when there are changes, you maintain an overview of the devices, objects, tags, faceplate types, alarms, scripts, etc. that you have used.
- From the cross-references, you can jump directly to the respective location of use of objects.
- You can see whether the respective object uses other objects or is itself used.
- During troubleshooting you will learn, for example, the following:
  - Which objects are used in which screen and on which devices.
  - Which alarms and parameter sets are displayed in which display.
  - Which tag is used in which alarm or which object.

See also

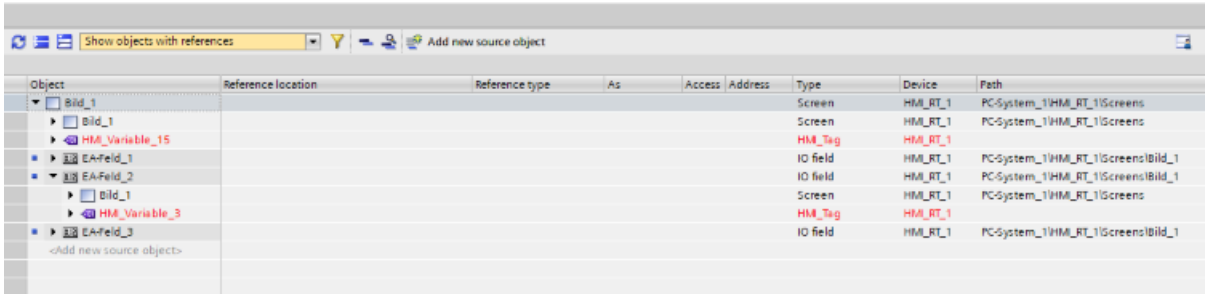
Displaying the "Cross-references" editor (Page 219)

Display cross-references in the Inspector window (Page 221)

3.13.2 Textual cross-references

Introduction

Textual cross-references are cross-references that are stored as pure text. With the help of textual cross-references, you cannot navigate to a different object. These cross-references are shown in red.



Textual cross-references are created as follows:

- You create a textual cross-reference for a non-existent object manually by entering the object name under Property and updating the cross-reference list.
- The object used has already been deleted, and the connection to it is shown as textual cross-reference.

Basics

When references and subordinate objects of a source object are deleted, they are converted into textual cross-references. A textual cross-reference is not stored when the source object itself is deleted.

Note

Almost all objects, with the exception of tag and alarm logs, support textual cross-references.

Note

Textual cross-references cannot be deleted.

You can also create a textual cross-reference manually to establish a connection to a non-existent object. This procedure can be helpful if you want to access tags directly using scripts.

Using textual cross-references when working with scripts

When working with scripts, you can reference tags directly with the help of textual cross-references.

When you enter an object name, the referenced object is searched for.

Sample code

```
export async function IO_Field_1_OnTapped(item, x, y, modifiers, trigger)
{
    HMIRuntime.Tags.SysFct.IncreaseTag("HMI_Tag_1", 0);
}
```

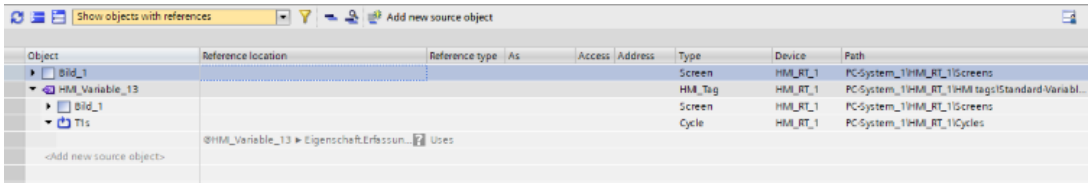
If a tag exists under the name "Variable_Test", it is referenced.

If there is no tag with the name "Variable_Test", the system creates a textual cross-reference for the tag.

The textual cross-reference is assigned to the tag that you create under this name. The textual cross-reference becomes a linked cross-reference.

3.13.3 Invalid cross-references

A cross-reference can become invalid as a result of certain changes to the configuration. Invalid cross-references are shown as grayed out.



Invalid cross-references do not have any negative implications for your project.

If an object or location of use is invalid, you cannot navigate to that object or location of use. You cannot change the properties of an invalid location of use. No shortcut menu is available for invalid locations of use.

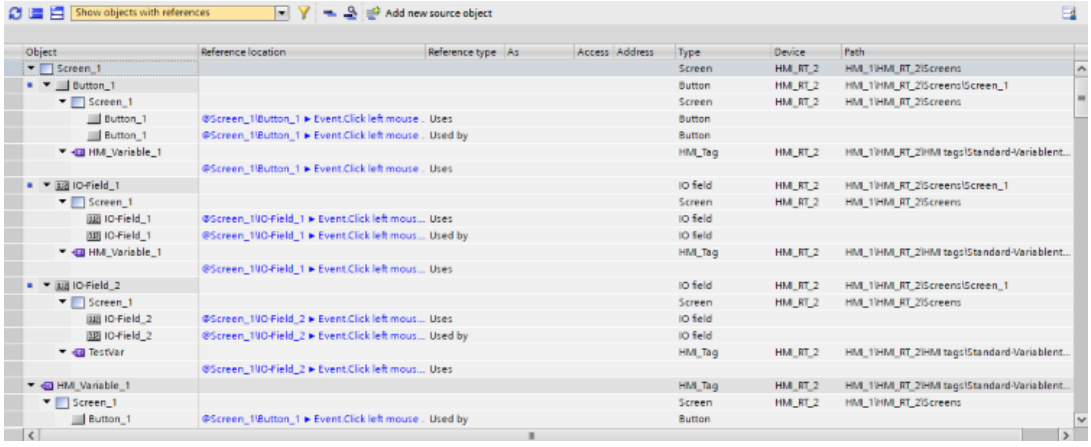
3.13.4 Displaying the "Cross-references" editor

Introduction

Depending on the object selected in the project tree, all the cross-references available for the object are displayed. In the project tree you can show cross-references for HMI devices, folders, and all editors. The detail view also lets you select individual objects of the editors, for example, individual screens.

Cross-references are always displayed for the selected object and for all lower-level objects.

The display of the cross-references for an object is specific to the project and device. You can display multiple cross-reference editors at the same time.



Requirement

- A project has been created.
- Multiple objects with references have been created.

Procedure

1. Select the required entry in the project tree or detail view.
2. Select "Cross-references" in the shortcut menu.
   Alternatively, select the "Cross-reference" command from the "Tools" menu.
   The "Cross-references" editor is opened in the working area.
   - The relevant selected object is the "source object" and is at the top in the "Object" column.
   - Point of use: Shows the locations at which the objects shown in the cross-reference list are used.
   - Type of use: Indicates whether the object is used.
3. To go to the location of use for a specific object, click on the cross-reference shown in blue in the "Point of use" column.
4. To go to the specific object, select the required object and select "Open editor" from the shortcut menu.
   The editor in which the object can be processed opens.
5. You can perform the following actions using the icons in the toolbar:
   - Updating
   - Collapse all
   - Expand all
   - Define filters for the cross-reference list
   - Show overlapping access
   - Check overlapping access
   - Add new source object
   - Save window settings

Note

Restoring cross-references after a project upgrade.

After a project upgrade to a higher TIA Portal version, you have the option of recreating the associated cross-reference data for the updated project.

This happens automatically at the end of the upgrade process. The updated project is opened and the cross-references are available again.

If the cross-reference data is inconsistent or incomplete, you will receive information. A banner text with a link to this information is displayed when the cross-reference list is opened. In this case, you can reorganize the cross-references yourself.

Delete

If you select a source object and "Delete" in the shortcut menu or press <Del>, then the object and not only the cross-reference is deleted.

A warning is displayed before the deletion.

After an object has been deleted, the cross-reference list is updated.

You cannot delete cross-reference lists.

3.13.5 Display cross-references in the Inspector window

Introduction

In the Inspector window, the cross-reference information for a selected object is displayed in the "Info > Cross-references" tab. In this way, you can see at a glance all the cross-references of the respective object without changing the cross-reference list.

All included elements and their use in the cross-reference list are displayed for structured tags, user data types and instances of a PLC data type.

Requirement

- A project has been created.
- Multiple objects with references have been created.

Procedure

1. Select an object.
2. Select "Cross-reference information" in the shortcut menu.
   The cross-references are opened in the Inspector window.

	**Tip for an efficient procedure**
If you select another object, the contents of the Inspector window are automatically refreshed. The cross-references to the selected object are displayed.	

Note

"Inspector window" restrictions

The Inspector window provides you with almost the same functions as in the "Cross-references" editor.

With more than 10,000 objects, the source objects and referenced objects are not sorted alphabetically in the Inspector window.

Example

When you select an object in the screen and an HMI tag is being used as the process tag at the object, the object and the linked HMI tag are displayed in the cross-references.

When the HMI tag is interconnected with a PLC tag or a data block tag, the locations of use of the interconnected PLC tag or data block tag are also displayed.

Result

The instances where and the other objects by which the selected object is being used are displayed.

The table below shows the additional information listed in the "About > Cross-reference" tab:

Column	Content/meaning
Object	Name of the object that uses the lower-level objects or that is being used by the lower-level objects.
Point of use	Each point of use, for example, an object or event
Type of use	Object relationships between the source object and its uses: • "Used by": The source object uses this object • "Uses": The source object is used by this object • "Type instance": The source object is a type or an instance of the referenced object • "Belongs to": The source object belongs to the reference object • "Contains": The source object contains the reference object
As	Shows additional information about objects, for example, that a tag is used by several devices
Access	Shows whether access to the object is read (R) and/or write (W)
Address	Displays the address of the object
Type	Displays information about the type and language used to create the object
Device	Displays the associated device name
Path	Displays the path of the object in the project tree with specification of folders and groups.
Comment	Displays user comments on individual objects, if available.

Depending on the installed products and the selected objects, additional columns or different columns are displayed for the cross-references.

3.13.6 Restoring cross-references after project upgrade

Introduction

After a project upgrade to a higher TIA Portal version, you have the option of recreating the associated cross-reference data for the updated project.

This happens automatically at the end of the upgrade process. The updated project is then opened and the cross-references are available again.

If the cross-reference data is inconsistent or incomplete, you will receive information. A banner text with this information and a further link is displayed when you display the cross-references.

In this case, you can reorganize the cross-references yourself.

Requirement

You have performed a project upgrade and received the message that the cross-reference data is inconsistent or incomplete.

Recreate cross-reference information

To create new cross-reference information, follow these steps:

1. Click the link to restore the cross-reference information in the displayed message.
   As an alternative, you can also navigate to "Cross-references" under "Tools > Settings > General".
2. Click the displayed "Recreate the cross-reference information" button.
3. Once the restore process is complete, check the message in the Inspector window to see if the operation was successful.
   If the process was not successful, you receive an error message.
4. In the cross-reference list, click the "Update" button to close the displayed banner again.

Result

The cross-references for the selected project have been recreated.

3.14 Configuring cycles

3.14.1 Basics of cycles

Cycles are used to control actions that reoccur regularly in runtime. Classic applications include:

- The acquisition cycle
- The logging cycle
- The screen cycle

You can also define your own cycles in addition to those already provided in WinCC.

Principle

Typical applications for cycles:

- Acquisition of external tags
  The acquisition cycle determines when the HMI device will read the process value of an external tag from the PLC. Set the acquisition cycle to suit the rate of change of the process values. The temperature of an oven, for example, changes much more slowly than the speed of an electrical drive.
  Do not set the acquisition cycle too low, since this will unnecessarily increase the communication load of the process.
- Triggering scheduled tasks
  In scheduled tasks you have the option to configure a task with a cyclical trigger. Use the cycle time to determine when the scheduled task is executed.
- Logging process values
  The logging cycle determines when a process value is saved in the logging database. The logging cycle is always an integer multiple of the acquisition cycle.

The smallest value for a cycle in Runtime Unified is 100 ms. You can configure all other values with an increment of 50 ms. The default value for the setting is 500 ms.

Application example

You can use cycles for the following tasks:

- To record and archive process values
- To trigger tasks
- To regularly log a process
- To draw attention to maintenance intervals

3.14.2 Defining cycles

Introduction

Use cycles to control actions that are run at regular intervals in runtime. You can also define your own cycles in addition to those already provided in WinCC.

Requirement

The project is open.

Procedure

To define a cycle, follow these steps:

1. Double-click the "Cycles" entry in the project navigation.
   The "Cycles" editor opens.
2. In the "Name" column of the "Cycles" editor, double-click "Add".
   A new cycle time is created.
3. Enter a unique name in the "Name" field.
4. Select the desired cycle unit.
5. Select the desired value for the cycle time.
   The available selection of values varies depending on the cycle unit selected.
6. As an option, you can enter a comment regarding the use of the cycle.
7. Save the project.

Result

The cycle you configured is created and beside the default cycles in WinCC for use during configuration.

3.15 Configuring in multiple languages

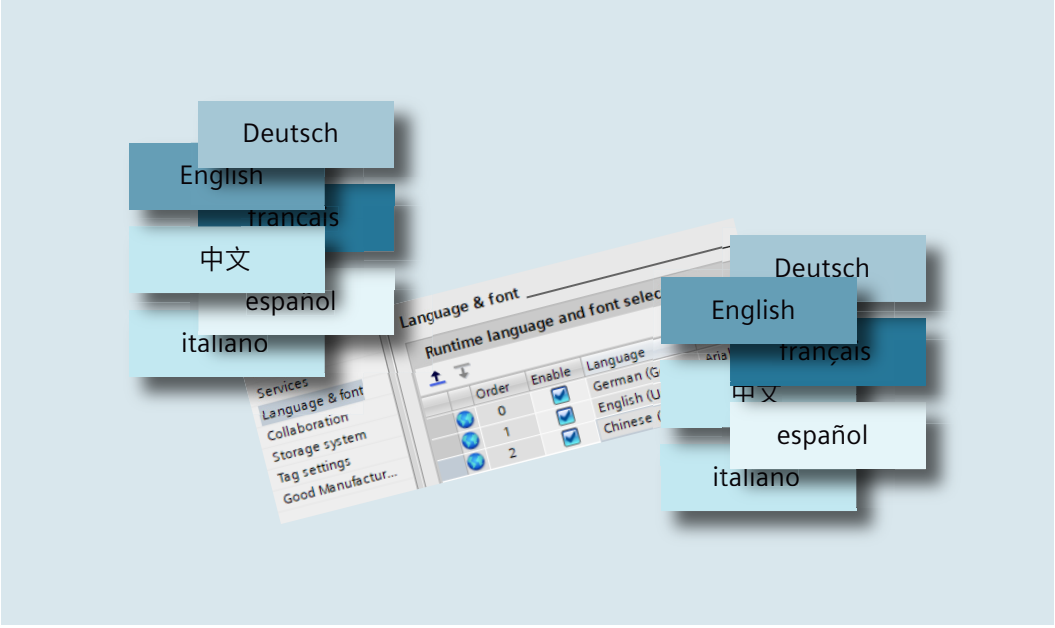
3.15.1 Languages in WinCC

User interface language and project languages

A distinction is drawn between two different language levels in WinCC:

- User interface language
  During configuration, the text in the WinCC menus and dialogs is displayed in the user interface language. The user interface language also affects the labeling of operating elements, the parameters of the system functions, the online help, etc.
- Project languages
  Project languages are all languages in which a project will later be used. Project languages are used to create a project in multiple languages.

The two language levels are completely independent of one another. For example, you can create English projects at any time using a German user interface and vice versa.



Project languages

The following languages are differentiated within the project languages:

- Reference language
  The reference language is the language that you use to configure the project initially. During configuration, you select one of the project languages as the reference language. You use the reference language as a template for translations. All of the texts for the project are first created in the reference language and then translated. While you are translating the texts, you can have them displayed simultaneously in the reference language.
- Editing language
  You produce translations of the texts in the editing language.
  Once you have created your project in the reference language, you can translate the texts into the remaining project languages. Select a project language respectively as an edit language and edit the texts for the appropriate language variant. You can change the editing language at any time.

  Note

  When switching the project languages, the assignment to the keys on the keyboard also changes. For some languages (for example, Spanish), the operating system does not allow you to switch to the corresponding keyboard assignment. In this case, the keyboard assignment is switched to English.

- Runtime languages
  Runtime languages are those project languages that are transferred to the HMI device. You decide which project languages to transfer to the HMI device depending on your project requirements.
  You must provide appropriate operator controls so that the operator can switch between languages in Runtime.

3.15.2 Settings for languages in the operating system

Introduction

The configuration PC operating system settings influence WinCC language management in the following areas:

- Selection of project languages
- Regional format of dates, times, currency, and numbers
- Displaying ASCII characters

Project language selection

A language is not available as a project language unless it is installed in the operating system.

Regional format of dates, times, currency, and numbers

WinCC specifies a fixed date and time format in the Date - Time field for the selected project language and runtime language.

In order for dates, times, and numbers to be presented correctly in the selected editing language, this language must be set in the Regional Options in the Control Panel.

Displaying ASCII characters

With text output fields, the display of ASCII characters as of 128 depends on the set language and the operating system being used.

If the same special characters are to be displayed on different PCs, the PCs must use the same operating system and regional settings.

3.15.3 Settings for Asian languages in the operating system

Settings on Western operating systems

If you want to enter Asian characters, you must enable support for this language in the operating system.

The Input Method Editor (IME) is available in Windows for configuring Asian texts. Without this editor, you can display Asian text but not edit it. For more information on the Input Method Editor, refer to the documentation for Windows. To enter Asian characters when configuring, switch to the Asian entry method in the "Input Method Editor".

Switch the operating system to the appropriate language to have language-specific project texts, such as alarm texts, displayed in the simulator in Asian characters.

Settings on Asian operating systems

If you are configuring on an Asian operating system, you must switch to the English default input language to enter ASCII characters, for example, for object names. As the English default input language is included in the basic installation of the operating system, you do not need to install an additional input locale.

Enabling language support

1. Open "Settings" from the Windows Start menu.
2. Select "Time & Language > Language".
3. Select "Preferred languages > Add a language".
4. Select the language.
   The "Install language features" dialog box is displayed.
5. Install the selected language pack and enable additional options, if necessary
6. Under "Related settings > Administrative language settings", you can find settings for programs that do not support Unicode.

Note

The options may be named or arranged differently depending on the operating system.

3.15.4 Setting project languages

3.15.4.1 Selecting the user interface language

Introduction

The user interface language is used for displaying menu entries, title bars, infotext, dialog texts and other designations in the WinCC user interface.

You can switch between the installed user interface languages during configuration. The labeling of the operating elements remains in the language you set when you added the object even if you change the user interface language.

Procedure

1. Select "Options > Settings" in the menu.
   The "Settings" dialog box is opened.
2. Select the user interface language under "General > General".

Result

WinCC will use the selected language as user interface language.

3.15.4.2 Enabling project languages

Introduction

The project languages are set in the "Project languages" editor. You define which project language is to be the reference language and which the editing language.

Enabling project languages

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The lower-level elements will be displayed.
2. Double-click on "Project languages".
   The possible project languages will be displayed in the working area.
3. Enable the relevant project languages.

Note

Copying multilingual objects

The copies of multilingual objects to a different project only include text objects in the project languages which are activated in the target project. Activate all project languages in the target project to include the corresponding text objects when transferring the copy.

Disabling project languages

1. Disable the languages which are not relevant for the project.

NOTICE
If you disable a project language, all text and graphic objects you have already created in this language will be deleted from the current project.

3.15.4.3 Selecting the reference language and editing language

Introduction

The project languages are set in the "Project languages" editor. You define which project language is to be the reference language and which the editing language. You can change the editing language at any time.

Requirements

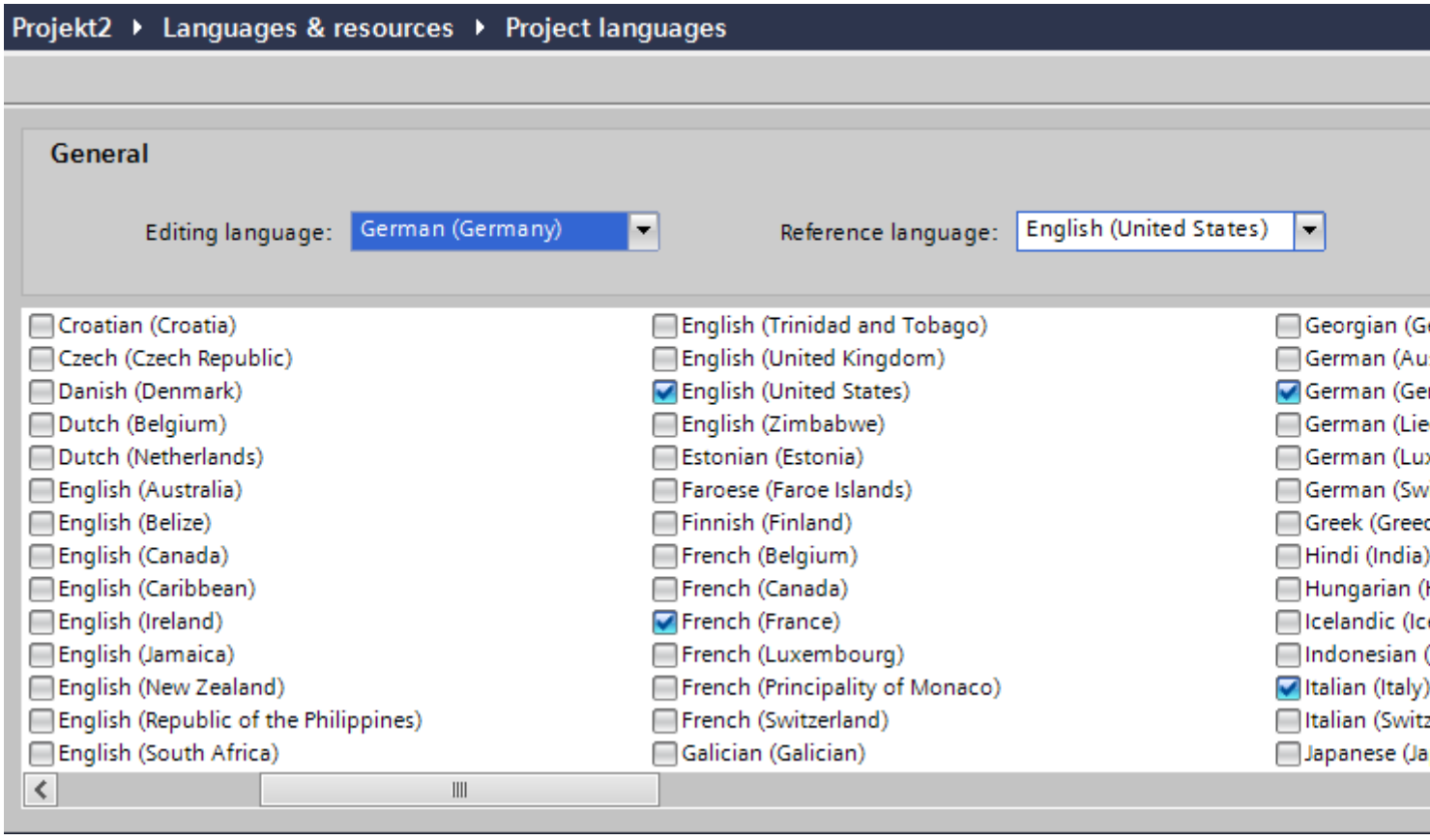
The "Project languages" editor is open.

Several project languages have been activated.

Selecting the reference language and editing language

1. Click the arrow in the drop-down list in the "General > Editing language" section.
2. Click the required language in the drop-down list, for example, German.
3. Click on the arrow in the drop-down list in the "General > Reference language" section.
4. Click the required language in the drop-down list, for example, English.

The language selection is displayed in the list box.



Result

You have now selected the editing and reference languages.

If you change the editing language, all future text input will be stored in the new editing language.

See also

Configuring multilingual alarm texts (Page 747)

3.15.5 Creating one project in multiple languages

3.15.5.1 Working with multiple languages

Multilingual configuration in WinCC

You can configure your projects in multiple languages using WinCC. There are various reasons for creating a project in multiple languages:

- You would like to use a project in more than one country.
  You would like to create the project in multiple languages. When the HMI device is commissioned, only the language spoken by the operators at the respective site is transferred to the HMI device.
- The operators of a system speak different languages.
  Example: An HMI device is used in Germany, but the operating personnel understand only English.

Translating project texts

With WinCC, you can enter project texts directly in multiple languages in various editors, for example, in the "Project texts" editor. WinCC also allows you to export and import your configuration for translation purposes. This is advantageous if you configure projects containing a large amount of text and want to have it translated.

Language management and translation in WinCC

The following editors are used to manage languages and translate texts in WinCC:

Editor	Short description
Project languages	Selection of project languages, editing language and reference language.
Languages and fonts	Selection of Runtime languages and the fonts used on the HMI device.
Project texts	Central management of configured texts in all project languages.
Project graphics	Project graphics for managing graphics and their language-specific versions.

See also

Configuring multilingual alarm texts (Page 747)

3.15.5.2 Basics of project texts

Texts in different languages in the project

Texts that are output on HMI devices during process operation are typically entered in the language in which the automation solution is programmed. Comments and the names of objects are also entered in this language.

If operators do not understand this language, they require a translation of all operator-relevant texts into a language they understand. You can therefore translate all the texts into any language. In this way, you ensure that anyone who is subsequently confronted with the texts in the project sees the texts in his/her language of choice.

User texts and system texts

In the interests of clarity, a distinction is drawn between user texts and system texts:

- User texts are texts created by the user.
- System texts are texts created automatically and which are a product of configuration in the project.

The project texts are managed in the project text editor. This can be found in the project tree under "Languages & Resources > Project texts".

Examples of multilingual project texts

You can create and manage the following text types in multiple languages:

- Display texts
- Alarm texts
- Comments in tables
- Labels of screen objects
- Texts in text lists

Translating texts

There are two ways of translating texts.

- Translating texts directly
  You can enter the translations for the individual project languages directly in the "Project texts" editor.
- Translating texts using reference texts
  You can change the editing language for shorter texts. You can enter the new texts in the editing language while the texts of the reference language are displayed.

Missing translation for texts of screen elements

Note

If no translation in the current user interface language exists for a text of a screen element, the text entered for the default language is displayed.

See also

Configuring multilingual alarm texts (Page 747)

3.15.5.3 Translating texts directly

Translating texts

If you use several languages in your project, you can translate individual texts directly. As soon as you change the language of the software user interface, the translated texts are available in the selected language.

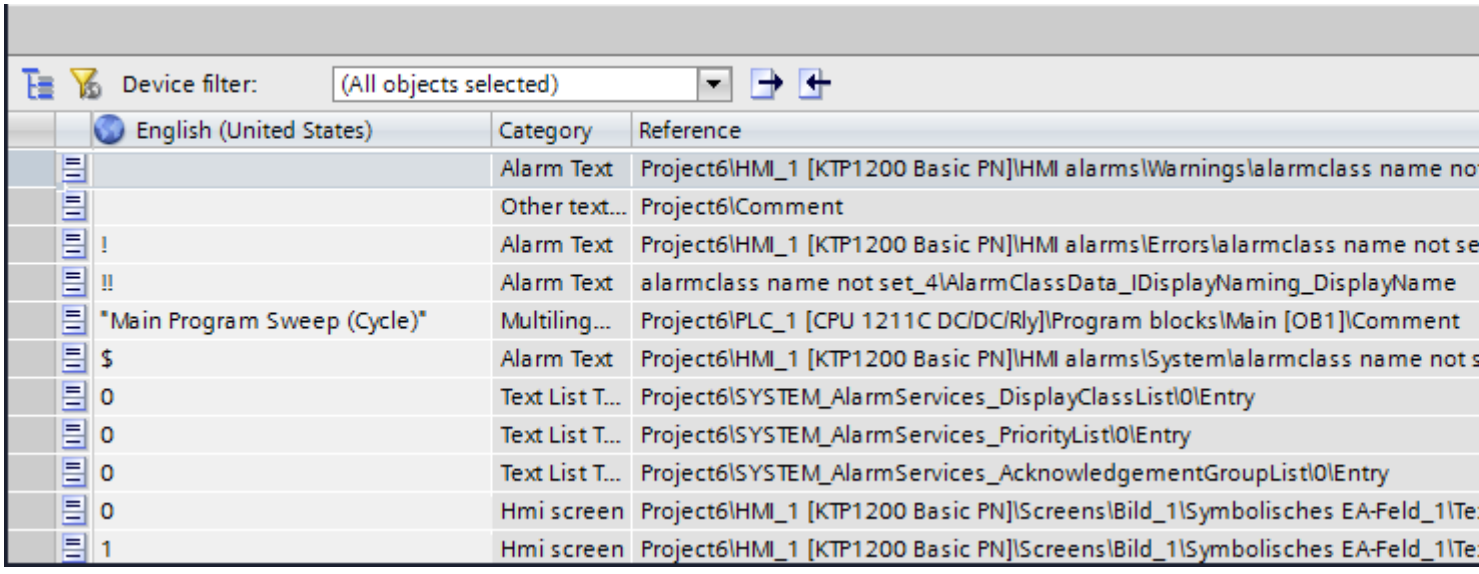
Requirements

- You are in the project view.
- A project is open.
- You have selected at least two other project languages.

Procedure

Proceed as follows to translate individual texts:

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The elements below this are displayed.
2. Double-click on "Project texts".
   A list with the texts in the project is displayed in the work area. There is a separate column for each project language.



3. To group identical texts and translate them simultaneously, click "" in the toolbar.
4. To hide texts that do not have a translation, click in the toolbar.
5. Click a cell and enter the translation.

Result

You have translated individual texts in the "Project texts" editor. The texts will then be displayed in the runtime language.

See also

Configuring multilingual alarm texts (Page 747)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

3.15.5.4 Translating texts using reference texts

Introduction

After changing the editing language, all texts are shown in input boxes in the new editing language. If there is not yet a translation available for this language, the input boxes are empty or filled with default values.

If you enter text again in an input field, this is saved in the current editing language. Following this, the texts exist in two project languages for this input field, in the previous editing language and in the current editing language. This makes it possible to create texts in several project languages.

You can display existing translations for an input box in other project languages. These serve as a comparison for text input in the current editing language and they are known as the reference language.

Requirement

There is at least one translation into a different project language for an input field.

Procedure

To display the translation of an input cell in a reference language, follow these steps:

1. Select "Tasks > Languages & resources" in the task card.
2. Select a reference language from the "Reference language" drop-down list.

Result

The reference language is preset. If you click in a text field, translations that already exist in other project languages are shown in the "Tasks > Reference language" task card.

3.15.5.5 Exporting project texts

Project texts are exported for translation. Texts are exported to Office Open XML files ending in ".xlsx". These can be edited in Microsoft Excel, for example.

You can exchange the file with the translators and import it directly back into the project after translation.

Requirements

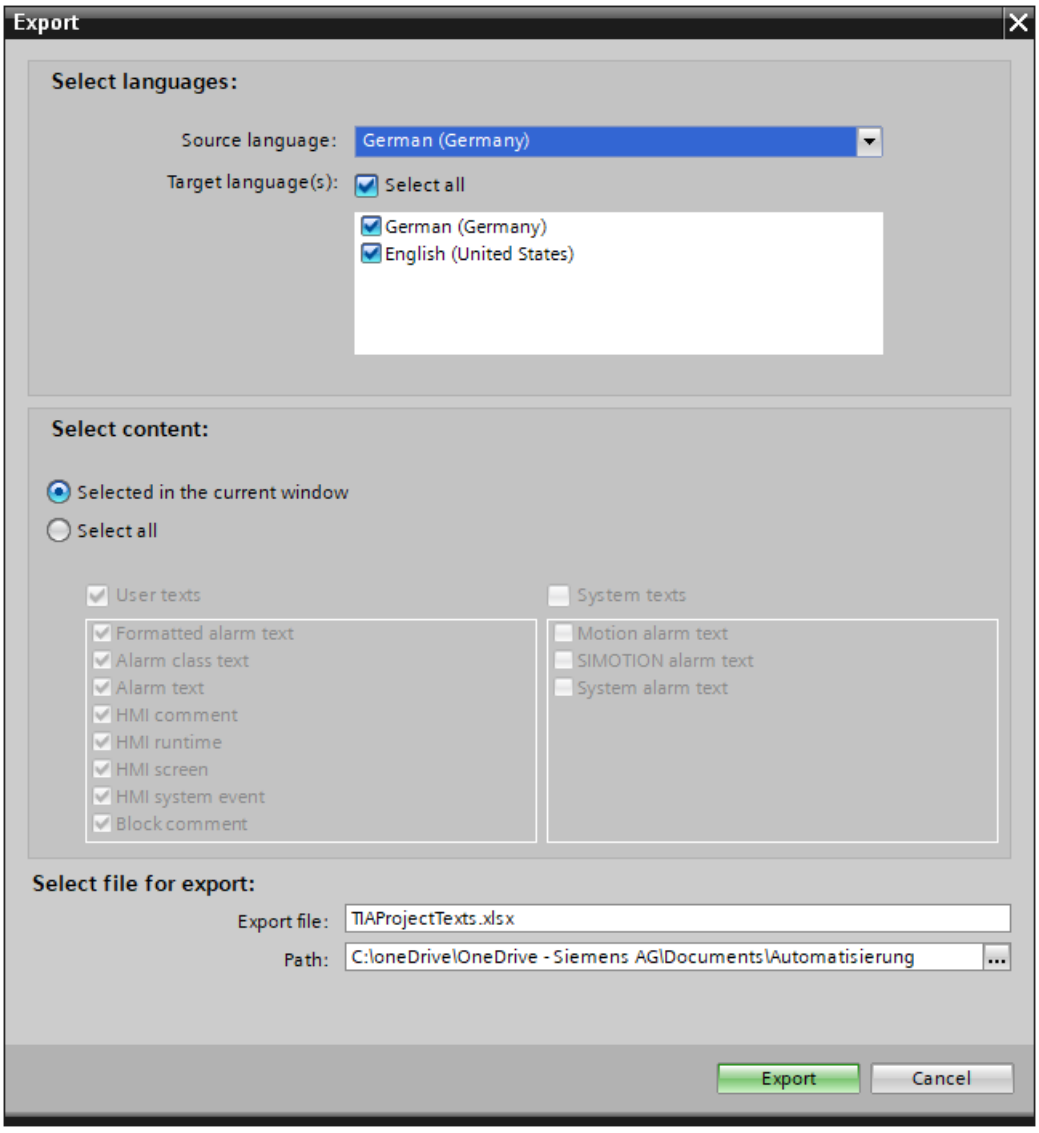
- At least two languages have been enabled in the "Project languages" editor, e.g. Italian and French.

Exporting project texts

To export individual project texts, proceed as follows:

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The child elements are displayed.
2. Double-click on "Project texts". The "Project texts" editor will open.
3. Select the texts you want to export.

4. Click on the button. The "Export" dialog box opens.



5. Select the language you want to translate from in the "Source language" drop-down list.
6. Select the check box(es) for the target language(s) into which you want the texts to be translated.
7. Under "Select content", select the categories of texts you want to translate.
8. In the "Export file" input field, specify the file name for the export file.
9. Enter a file path for the export file in the "Path" input field.
10. Click "Export".

Result

The texts selected in the "Project texts" editor are written to an xlsx file. The xlsx file will be stored in the specified folder.

Note

Project texts in library objects cannot be exported.

See also

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

3.15.5.6 Importing project texts

Once translation is complete, import the xlsx file with the translated texts. The target languages are imported to the corresponding object in the project.

Note

In WinCC, you import previously exported project texts only into the source project. Importing into a different project is not supported.

Requirements

- At least two languages have been enabled in the "Project languages" editor.

Importing project texts

To import a file with project texts, proceed as follows:

1. Click on the arrow to the left of "Languages & resources" in the project tree.
   The lower-level elements will be displayed.
2. Double-click on "Project texts".
   The "Project texts" editor will open.
3. Click on the  button. The "Import" dialog box opens.
4. Select the path and the file name of the import file from the "Select file for import" field.
5. Activate the "Import source language" check box if you have made changes to the source language in the export file and would like to overwrite the entries in the project with the changes.
6. Click on "Import".

Result

You have imported the project texts.

See also

Configuring multilingual alarm texts (Page 747)

3.15.6 Using language-specific graphics

3.15.6.1 "Project graphics" editor

Introduction

You use the "Project graphics" editor to manage the configured graphic objects in different language versions. Multilingual projects require language-dependent versions of the graphics, for example, if:

- The graphics contain text
- Cultural aspects play a role

Opening the "Project graphics" editor

- Double-click in the project tree on "Languages and resources > Project graphics".

Work area

The work area displays all configured graphic objects in a table. There is a separate column in the table for each project language. Each column in the table contains the versions of the graphics for one particular language.

In addition, you can specify a default graphic for each graphic to be displayed whenever a language-specific graphic for a project language does not exist.

Preview

The preview shows you how the graphics will look on various HMI devices.

3.15.6.2 Storing an image in the project graphics

Introduction

You use the "Graphics" editor to import graphics for use in screen objects in the "Screens" editor. Here you import and manage language-dependent graphic versions. A preview shows how the graphic looks on various HMI devices.

Note

File names for language-dependent graphics

Case is relevant in the file names of language-dependent graphics. Make sure to use the same spelling for all languages.

Note

File format of language-dependent graphics

In the language-dependent versions of a graphic, use only graphic files with the same format. Graphic versions with different file formats are not supported.

Requirement

- The language-dependent versions of a graphic are available.
- Multiple languages have been enabled in the "Project languages" editor.
- The "Graphics" editor is open.

Adapting the view of the project graphics

1. To show/hide the columns with language-dependent graphics, click on  .
2. To add another project language to the table, click on  .

Inserting graphics

1. Click "<Add>" in the "Project graphics" table.
   The dialog for selecting a file opens.
2. Select the graphic file and click "Open."
   The graphic is inserted into the project as a standard graphic and displayed in all cells of this row in the "Graphics" editor.
   The standard graphic is displayed in Runtime for languages for which there is no language-specific graphic.
3. Right-click in the cell of a language for which a language-dependent version of this graphic is to be imported.

4. Select "Replace with graphic" in the shortcut menu.
   The dialog for selecting a file opens.
5. Select the graphic file and click "Open."
   The language-dependent version of the graphic is inserted in the table.

Alternatively, you can drag&drop a graphic from Windows Explorer to the desired position in the "Project graphics" table.

Inserting graphics by copying

1. Select "Copy" in the shortcut menu of an existing graphic.
2. Select "Paste" in the shortcut menu of an empty cell.
   A new row is inserted. The copied graphic is inserted for all languages.
3. Select "Paste" in the shortcut menu of a filled cell.
   The existing graphic is replaced by the copied graphic.

Displaying graphics in the device preview

1. Select an HMI device in the "Select device for preview" drop-down list in the editor.
2. Click on a graphic in the table.
   In the Inspector window under "Properties > General > Preview for HMI device", you see the graphic as it will appear in Runtime on the selected HMI device.

Result

The inserted graphics are available in the project. The graphic assigned to the respective editing language will be displayed during editing. The standard graphic is displayed in editing languages for which no screen has been imported.

The screens assigned to the respective Runtime language are displayed in Runtime. The standard graphic is displayed in Runtime languages for which no screen has been imported.

Note

If you disable a project language, all previously inserted graphics in this language are deleted in the current project.

3.15.6.3 Storing an external image in the project graphics

Introduction

To use a graphic created in an external graphics program in screens, you add it to the project graphics.

Requirement

- Multiple languages have been enabled in the "Project languages" editor.
- The "Graphics" editor is open.
- There is a graphic in the "Graphics" editor.

Creating and adding a new graphic as an OLE object

1. Right-click in a cell containing a graphic.
2. Select "Replace with object" in the shortcut menu.
   The "Insert object" dialog box opens.

   Note

   In addition, the "External application running..." dialog opens. The dialog is closed when the external application has finished.

3. Select "Insert object > Create new" and an object type in the dialog.
4. Click "OK."
   The graphics program assigned by the operating system opens.
5. Create the graphic.
6. Close the graphics program.
   The graphic is saved in the standard format of the graphics program and inserted into the project graphics.

Result

The inserted OLE objects are available in the "Graphics" editor.

Versions of the graphics for the current editing language are displayed in the "Screens" editor. The default graphic is displayed in all editing languages for which no screen has been imported.

The graphic is displayed in Runtime in the set Runtime language. The default graphic is displayed in all Runtime languages for which no graphic has been imported.

3.15.6.4 Editing a graphic

Graphics can be edited directly from the project graphics with an external graphics program.

Requirement

- The "Graphics" editor is open.
- There is a graphic in the "Graphics" editor.

Editing a graphic

1. Select "Edit" in the shortcut menu for a graphic.
   The assigned program, e.g. "Paint", opens.

   Note

   In addition, the "External application running..." dialog opens. The dialog is closed when the external application has finished.

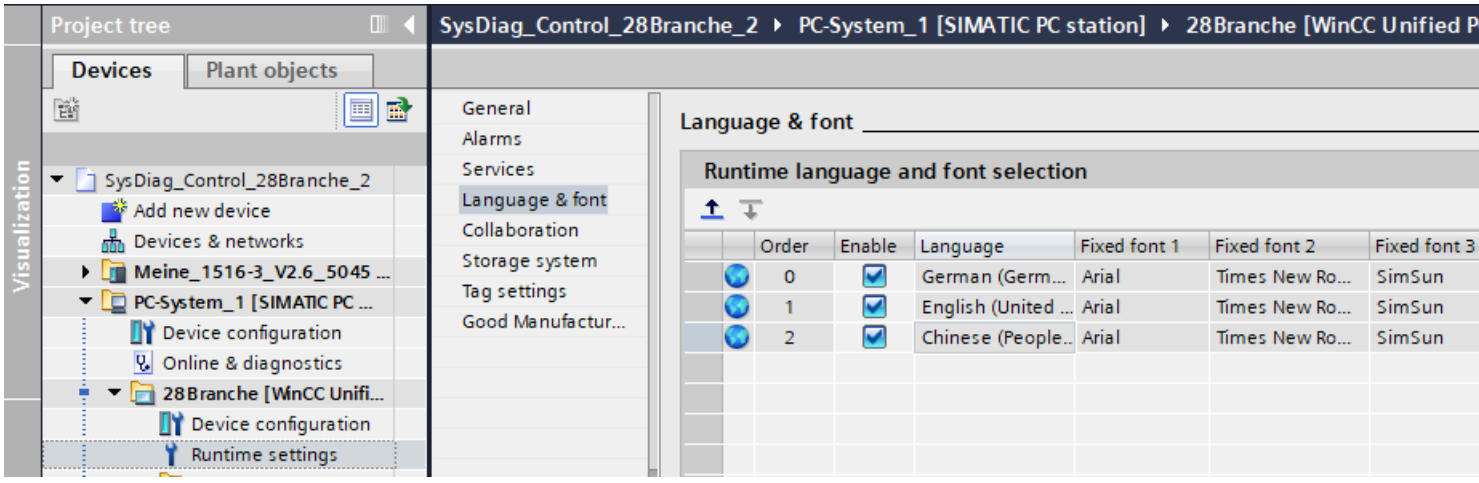
2. Edit the graphic.
3. Close the external program, e.g. with the "Close and return to document" command.
   The edited graphic is available in the project graphics.

3.15.7 Languages and fonts in runtime

3.15.7.1 Using multiple runtime languages

In the Runtime settings, you define which project languages are used in Runtime on a particular HMI device. The number of Runtime languages that are available at one time on the HMI device depends on the device. To enable the operator to switch between languages in Runtime, you need to configure a corresponding operator control.

1. In "Languages & Resources", you configure project languages that are available as Runtime languages for the respective device.
2. In "Runtime settings > Language & Font", you define the order in which the languages are switched.



At Runtime start, the Runtime project is displayed in the language that is set in the "User login" dialog. If this language is not configured in the Runtime settings of the HMI device, the language with the lowest number in the "Order" column is used.

You change the order with . Four predefined fonts are stored for each language. The fixed font 1 is always provided for the respective HMI device.

3.15.7.2 Own fonts

Downloading your own font

Using fonts that require licenses on HMI devices

When a font that requires a license is installed on your configuration PC, you can load this font onto your HMI device under certain conditions. Whether or not you can load the font onto your HMI device depends on the embeddability of the font.

Check the properties of the font in Windows:

1. Open the "C:\Windows\Fonts" folder in Windows Explorer.
2. Select the desired font.
   If several fonts are grouped into a font family, click on the entry for the font family. The individual fonts are now displayed.
3. Right-click the entry for the font.
4. Select "Properties".
5. In the "Details" tab, you find information about the embeddability of the font and the license terms and conditions.

The embeddability of fonts can have the following statuses:

- "Installable" status: Font can be installed on your HMI device without an additional license.
- "Editable" status: Font can be installed on your HMI device without an additional license.
- "Preview/Print" status: Warning during compilation that the font cannot be installed on your HMI device.
- "Restricted license embedding" status: Font can be installed on your HMI device with an additional license.
- "Bitmap embedding only" status: Compilation of the project is aborted with an error; the font cannot be installed on your HMI device.
- "No subsetting" status: Warning during compilation that the font can only be loaded onto the HMI device as a complete set. Substitute fonts and options for partial fonts cannot be used during compilation.

Note

The fonts configured in the Engineering System (TrueType fonts) are converted into web fonts and both TrueType fonts and web fonts are downloaded to the Runtime machine: WinCC Unified PC (Windows) or Unified Comfort Panel (Linux). The user is responsible for ensuring that the license of the fonts used allows this.

Additional information is available under Font redistribution (https://docs.microsoft.com/en-us/typography/fonts/font-faq#document-embedding) and OpenType_fsType (https://docs.microsoft.com/en-us/typography/opentype/spec/os2#fstype).

Loading fonts onto the HMI device

Introduction

When a font that requires a license is installed in Windows on your configuration PC, you can also use this font on your HMI device under certain conditions. You are responsible for ensuring that your devices have valid licenses.

Downloading a customer-specific font

The TIA Portal project does not include any configured TrueType or Web fonts. The Runtime font conversion uses an RDF filed that was written in the RDF language (Resource Description Framework) and is used for displaying information on the resources. The RDF file contains metadata in a structured format; the metadata comprise the font file (ttf) as CSD resource.

The result of the GetNearestFont mechanism is transferred to the Runtime. The GetNearestFont function first searches for the font of the same name. If such a font does not exist, other fonts are checked one after the other.

Note

The fonts configured in the Engineering System (TrueType fonts) are converted into web fonts and both TrueType fonts and web fonts are downloaded to the Runtime machine: WinCC Unified PC (Windows) or Unified Comfort Panel (Linux). The user is responsible for ensuring that the license of the fonts used allows this.

Additional information is available under Font redistribution (https://docs.microsoft.com/en-us/typography/fonts/font-faq#document-embedding) and OpenType_fsType (https://docs.microsoft.com/en-us/typography/opentype/spec/os2#fstype).

3.15.7.3 Methods for language switching

Introduction

To enable switching between configured Runtime languages on the HMI device, you provide a language switching means. This is necessary to enable the operator to switch between the various Runtime languages.

Methods for language switching

You can configure the following methods for language switching:

- Direct language selection
  Each language is set by means of a separate button. You create a button for each Runtime language.
- Language switching
  The operator switches the languages using a button. Each click of the button changes the Runtime language to the next Runtime language.

Regardless of the method used, the button names must be translated into each of the languages used. You can also configure an output field that displays the current language setting.

3.15.7.4 Starting a project in a different language

Introduction

If the language "German" is configured in Unified Comfort Panel, the Runtime project is always started in German, regardless of the defined language sequence.

To start the project in another language, use the "HMIRuntime.Language" system function.

Procedure

Proceed as follows to start a project once automatically in English:

1. Create a "Bool" type tag, for example, "doOnlyAtStartup".
2. For the project start screen, link the "Loaded" event with the following script:

```
export function Screen_1_OnLoaded(item)
{
if(!Tags('doOnlyAtStartup').Read())
{
HMIRuntime.Language = 1033;
Tags('doOnlyAtStartup').Write(true);
}
}
```

To start the project in another language, enter the decimal Windows Locale Code (LCID=Locale ID) for the desired language instead of "1033". You can find the LCIDs on the Internet, for example.

3.15.7.5 Enabling the runtime language

Introduction

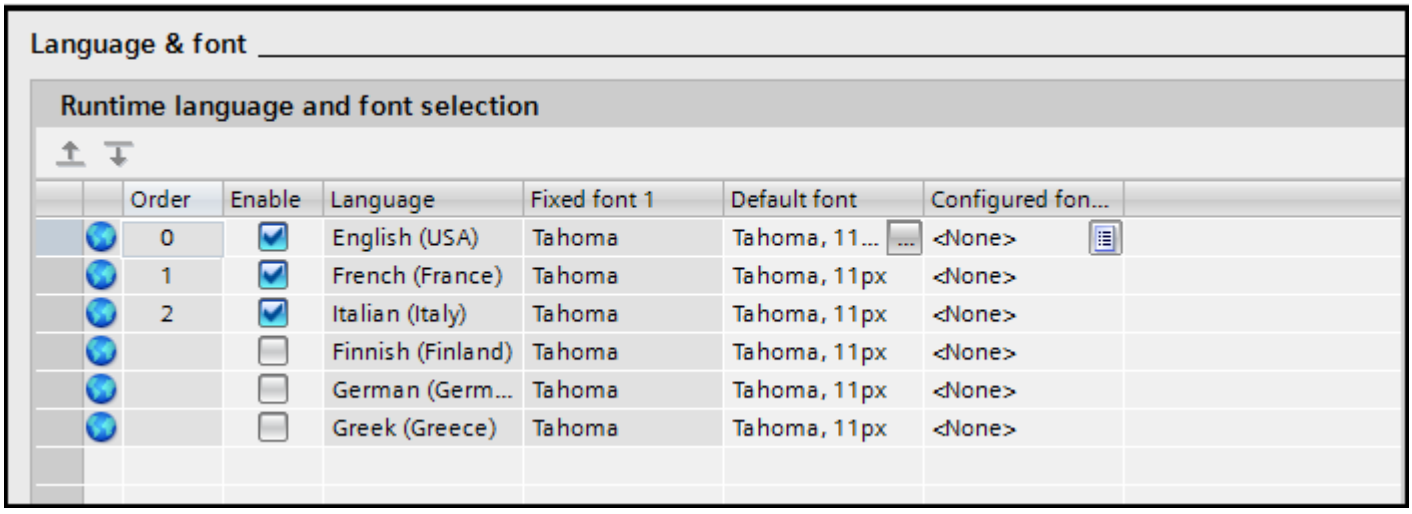
The "Language & Font" editor shows all project languages available in the project. You select which project languages are to be available as Runtime languages on the HMI device.

Requirements

Multiple languages have been selected in the "Project languages" editor.

Procedure

1. Double-click on "Runtime settings" in the project tree.
2. Click on "Language & Font".
3. Select the following languages:
   - English
   - French
   - Italian



Result

You have now set three Runtime languages. A number is automatically assigned to each language in the "Order" column. The enabled Runtime languages are loaded onto the HMI device with the compiled project.

If the number of languages selected exceeds the number that can be transferred to the HMI device, the table background changes color.

3.15.7.6 Standardizing font for all languages

Introduction

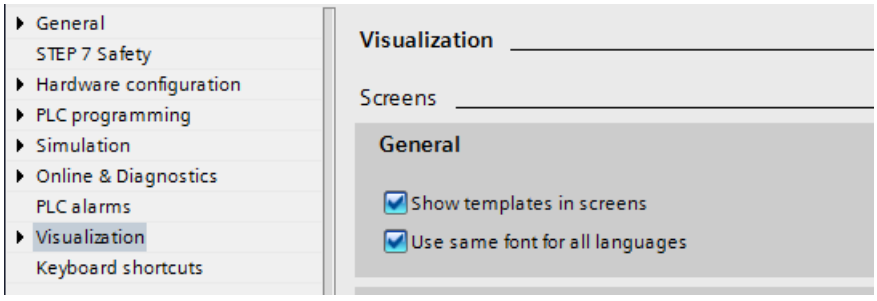
You can standardize the font for all project languages during configuration with the "Use same font for all languages" option.

Requirement

- Multiple languages have been selected in the "Project languages" editor.
- Multiple languages have been selected in the "Language & font" editor.
- The same font is defined for the selected runtime languages under "Configured font".

Procedure

1. In the "Options > Settings > Visualization > General" menu, select the "Use same font for all languages" option.



Result

You have enabled the option "Use same font for all languages". If you change the font of an object in one language during configuration, this font will be applied to all active languages.

3.15.7.7 Specific features of Asian and Eastern languages in runtime

Introduction

Note the following special considerations for the operation in Runtime of projects for Asian languages.

Note

During configuration, only use the Asian fonts that your configuration computer supports.

Memory requirement for Asian character sets

The memory requirement is greater when using Asian languages. Therefore look out for corresponding error messages when compiling the project.

Font size for Asian character sets

Use at least a font size of 10 points to display the text of projects created for Asian languages in Runtime. Asian characters will become illegible if smaller font sizes are used. This also applies to the default font in the Runtime settings under "Language & font".

3.16 Performance features

3.16.1 General technical data

3.16.1.1 SIMATIC Unified Comfort Panel

Unified Comfort Panel

The following tables of performance features help you to assess whether your project conforms to the system limits of a given HMI device.

The specified maximum values are not additive. It cannot be guaranteed that configurations running on the devices at the full system limits will be functional.

Furthermore, the complexity of configuring the screens, such as the number of objects per screen, the number of tag connections, cycle times and scripts, has a significant influence on the open screen times and the performance in runtime.

In addition to the specified limits, allowances must be made for restrictions imposed by configuration memory resources.

Tags

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of tags in the project	8000	
Number of elements per array	1600	

Alarms

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of alarm classes	32	
Number of discrete alarms	9000	
Number of analog alarms	300	

	Unified Comfort 7-12"	Unified Comfort 15-22"
Size of the alarm buffer [1)]	2000	
Length of an alarm in characters	512	
Number of alarm texts per alarm	10	
Number of process values per alarm	10	
Number of queued alarm events	750	
Number of controller alarms	160000	
Number of OPC UA A&C alarms	20000	

1) Corresponds to the number of all states of the messages of all configured alarm classes and includes the alarms of alarm classes which are not shown in an alarm view due to the configuration.

Number of alarms that can be displayed in an alarm view

The maximum number of alarms that can be displayed in runtime depends on the selected view.

	Unified Comfort 7-12"	Unified Comfort 15-22"
Show active alarms	No restriction	
Show defined alarms	No restriction	
Alarm statistics - view	No restriction	
Show logged alarms	1000	
Show and update logged alarms	100	

Screens

	Unified Comfort 7-12"	Unified Comfort 15-22"
Maximum size in the engineering system	20,000 * 20,000 pixels	
Maximum size in runtime	20,000 * 20,000 pixels	
Number of screens	1200	
Number of lower-level screen windows	10	
Number of objects per screen	800	1200
Number of objects from the "Controls" area per screen	40	80
Number of tags per screen	600	800

Parameter sets

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of parameter set types	750	
Number of parameter set type elements	1000	

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of parameter sets	2000	
Reserved memory for data records in the internal Flash	12 MB	

Libraries

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of versions of dynamic SVGs	1000 with an average size of 10 KB	

Logs

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of logs	50	
Maximum size of a log	4 TB	
Maximum size of a segment	4 TB	
Number of logging tags:	SQLite: 5000	
Memory requirements of the data log		
Size of entry of logging tag	The size of the entry of a logging tag is largely determined by the data type. Depending on the data type, the following memory requirements apply: • 32-bit value, e.g. Bool, Int, LReal, ... : ~ 80 byte / entry • 64-bit value, e.g. LInt, DateTime, LTime, ... ~ 106 byte / entry • Text value (any length), e.g. WString, WChar: ~ 586 bytes/entry	
Additional memory requirement of a segment:	SQLite: Approx. 0.5 MB	
Memory requirements of the alarm log		
Basic entry in the alarm log without alarm text:	SQLite: Approx. 300 bytes	
Memory requirement of the alarm text per character and language:	SQLite: at least 1 byte	
Additional memory requirement per language (one-off):	SQLite: Approx. 100 bytes	

Trends

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of trends	600	
Number of trends per trend control	20	
Number of trend areas per trend control	2	5

Text lists and graphics lists

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of graphics lists	750	
Number of text lists	750	
Number of entries per text or graphics list	750	
Number of graphic objects	6000	
Number of text elements	60000	

Scripts

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of scripts	600	
Number of functions per function list	25	

Scheduler

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of tasks, time- or event-triggered	70	

Communication

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of S7 connections	16	

Reporting

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of templates	--	
Number of report tasks	--	
Number of report tasks started at the same time	--	
Number of reports executed at the same time	--	

OPC UA

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of connected OPC UA clients	3	

Languages

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of runtime languages	32	

User management

	Unified Comfort 7-12"	Unified Comfort 15-22"
Number of roles	50	
Number of predefined function rights	20	
Number of users	200	

Project

	Unified Comfort 7-12"	Unified Comfort 15-22"
Size of the project files on the device	< 100 MB	

3.16.1.2 SIMATIC Unified PC

Unified PC based

The following tables of performance features help you to assess whether your project conforms to the system limits of a given HMI device.

The specified maximum values are not additive. It cannot be guaranteed that configurations running on the devices at the full system limits will be functional.

Furthermore, the complexity of configuring the screens, such as the number of objects per screen, the number of tag connections, cycle times and scripts, has a significant influence on the open screen times and the performance in runtime.

In addition to the specified limits, allowances must be made for restrictions imposed by configuration memory resources.

Tags

	SIMATIC Unified PC
Number of PowerTags	600000 (depends on the license)
Number of internal tags	200000
Number of elements per array	2000

Alarms

	SIMATIC Unified PC
Number of alarm classes	32
Number of discrete alarms	200000
Number of analog alarms	10000
Length of an alarm in characters	512
Number of alarm texts per alarm	10
Number of process values per alarm	10
Number of alarms for every second (continuous load)	20
Number of queued alarm events	unlimited
Number of alarms for every 10 seconds (alarm burst)	8000
Number of controller alarms	160000
Number of OPC UA A&C alarms	20000

Number of alarms that can be displayed in an alarm view

The maximum number of alarms that can be displayed in runtime depends on the selected view.

	SIMATIC Unified PC
Show active alarms	No restriction
Show defined alarms	No restriction
Alarm statistics - view	No restriction
Show logged alarms	1000
Show and update logged alarms	100

Screens

	SIMATIC Unified PC
Maximum size in the engineering system	20,000 * 20,000 pixels
Maximum size in runtime	20,000 * 20,000 pixels
Number of screens	2000
Number of lower-level screen windows	unlimited
Number of objects per screen	1500
Number of tags per screen	1000

Parameter sets

	SIMATIC Unified PC
Number of parameter set types	1000
Number of parameter set type elements	1000
Number of parameter sets	5000

Libraries

	SIMATIC Unified PC
Number of versions of dynamic SVGs	1000 with an average size of 10 KB

Logs

	SIMATIC Unified PC
Number of logs	100
Maximum size of a log	4 TB
Maximum size of a segment	4 TB
Number of logging tags:	SQLite: 5000 Microsoft SQL: maximum number of PowerTags
Number of entries in the data log per second with Microsoft SQL	Microsoft SQL: 30000
Memory requirements of the data log	
Size of entry of logging tag	The size of the entry of a logging tag is largely determined by the data type. Depending on the data type, the following memory requirements apply: • 32-bit value, e.g. Bool, Int, LReal, ... : ~ 80 byte / entry • 64-bit value, e.g. LInt, DateTime, LTime, ... ~ 106 byte / entry • Text value (any length), e.g. WString, WChar: ~ 586 bytes/entry
Additional memory requirement of a segment:	SQLite: Approx. 0.5 MB Microsoft SQL: Approx. 5 MB
Memory requirements of the alarm log	
Basic entry in the alarm log without alarm text:	SQLite: Approx. 300 bytes Microsoft SQL: Approx. 2000 bytes
Memory requirement of the alarm text per character and language:	SQLite: at least 1 byte Microsoft SQL: at least 2 bytes
Additional memory requirement per language (one-off):	SQLite: Approx. 100 bytes Microsoft SQL: Approx. 200 bytes
Additional memory requirement of a segment:	Microsoft SQL: Approx. 3.5 MB

Trends

	SIMATIC Unified PC
Number of trends	1000
Number of trends per trend control	60
Number of trend areas per trend control	5

Text lists and graphics lists

	SIMATIC Unified PC
Number of graphics lists	1000
Number of text lists	2000
Number of entries per text or graphics list	3500
Number of graphic objects	unlimited
Number of text elements	unlimited

Scripts

	SIMATIC Unified PC
Number of scripts	unlimited
Number of functions per function list	50

Scheduler

	SIMATIC Unified PC
Number of tasks, time- or event-triggered	200

Communication

	SIMATIC Unified PC
Number of S7 connections [1)]	128

[1)] SIMATIC NET is required to use more than 8 connections.

Reporting

	SIMATIC Unified PC
Number of templates	500
Number of report tasks	500
Number of report tasks started at the same time	20
Number of reports executed at the same time	5

OPC UA

	SIMATIC Unified PC
Number of connected OPC UA clients	10

Languages

	SIMATIC Unified PC
Number of runtime languages	32

User management

	SIMATIC Unified PC
Number of roles	50
Number of predefined function rights	20
Number of users	200

Plant objects

	SIMATIC Unified PC
Number of plant object types	400
Number of plant object instances	65000
Number of hierarchy levels	unlimited

3.16.2 Permitted special characters

Introduction

The following table shows the restrictions that must be observed when allocating names.

Permitted characters

Name	Restriction
Device name	The following constraints apply to the assignment of the device name: • Do not use the following characters: – , ; : ! ? " ' ^ ´ ` ~ _ + = / \ ¦ @ * # $ % & § ° ( ) [ ] { } < > – Spaces • Use upper case only. • The first character must be a letter. • The first 12 characters of the device name must be unique.
Object names	The use of the following special characters is not supported: • Pipe | • Forward slash / • Inverted slash \ • Dot . • Comma , • Semicolon ; • Colon : • Quotes " • Apostrophe ' • Angle brackets < > • Tilde ~ • Hash # • Dollar $ • Star * • Question mark ? The use of the following control characters is not supported: • \x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F When creating scripts, also consider the restrictions relating to special characters of the programming language.
User name	The use of the following special characters is not supported: • Forward slash / • Comma , • Parenthesis { }

Configuring screens

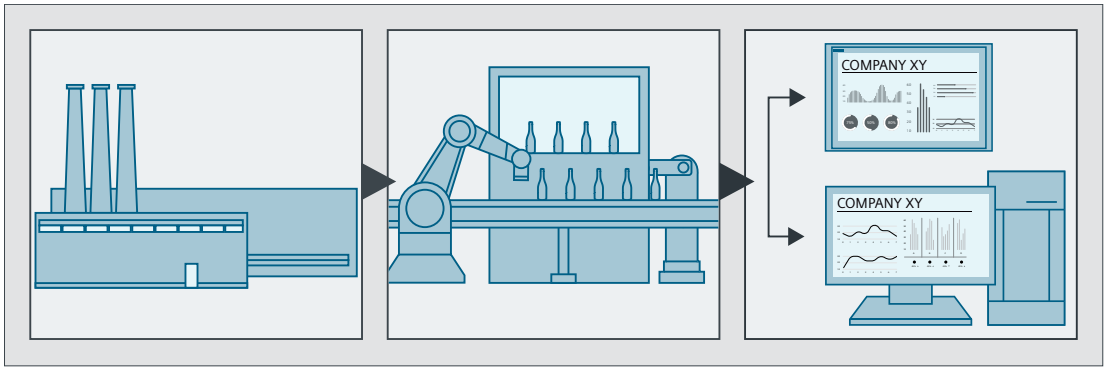
4

4.1 Basics

4.1.1 Basics of screens

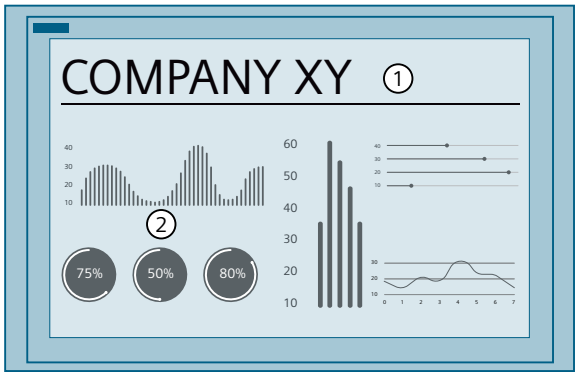
Introduction

In WinCC you create screens that a user can use to control and monitor machines and plants. When you create screens, the pre-defined object templates support you in visualizing your plant, displaying processes and specifying process values.



Structure of screens

Insert the objects you need to represent the process into your screen. Configure the objects to match the requirements of your process.



A screen can consist of static and dynamic elements:

- ① Static elements such as text or graphic objects in the screen above do not change in runtime.
- ② Dynamic elements change their status based on the process. You visualize current process values from the memory of the PLC or the HMI device. Dynamic objects include, for example, alphanumeric displays, trends and bars, as well as input fields on the HMI device, such as IO fields, switches and sliders. Process values and operator inputs are exchanged between the PLC and the HMI device by means of tags.

Start screen

The start screen is the initial screen displayed when the project is started in runtime. From the start screen, the operator can navigate to the other screens.
To compile and download a project, a screen must be defined as the start screen in the project.

Screen window

You display other project screens in the screen window. The screen window enhances navigation between screens and allows "screen in a screen" display.

You can use screen windows for purposes such as:

- Frequent switching between plant units
- Showing and hiding screens, for example without exiting central process visualization
- Displaying multiple plant units in one screen

4.1.2 Changing the screen resolution

Introduction

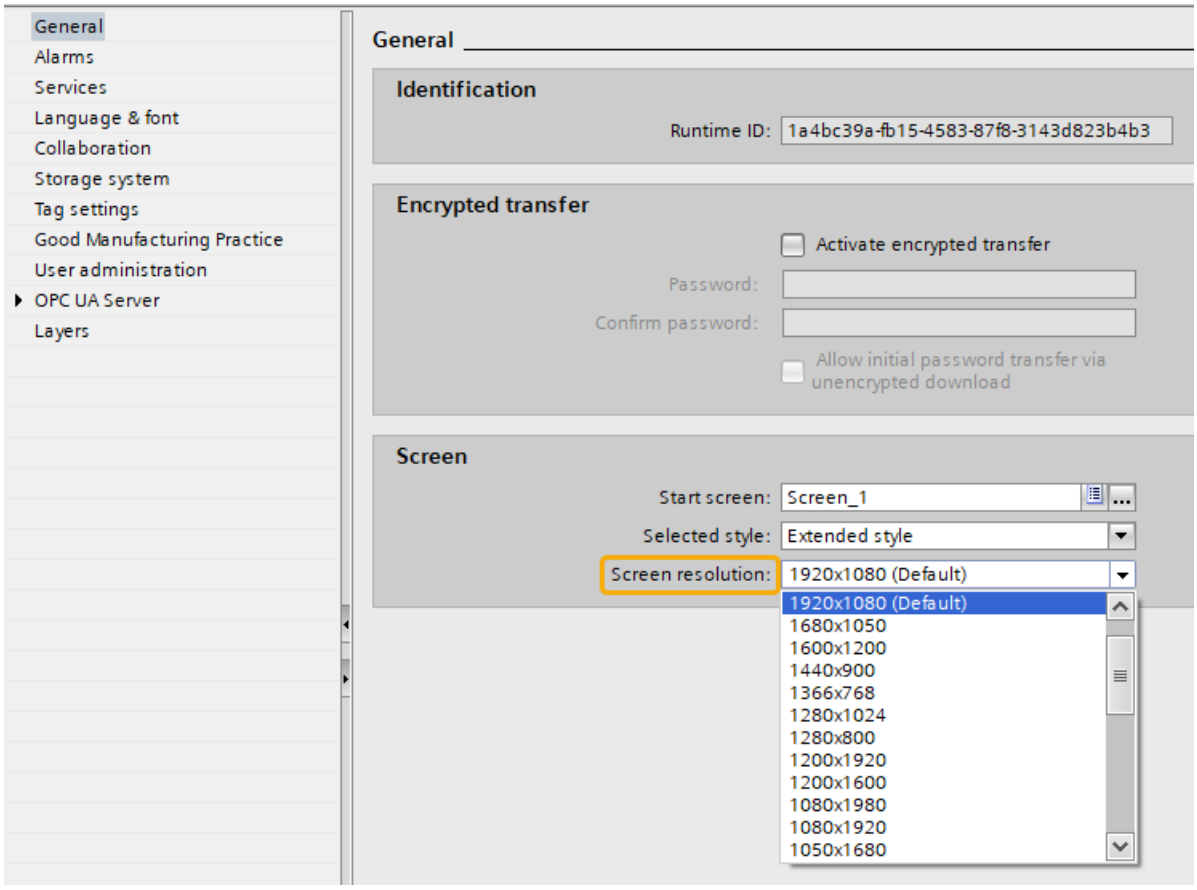
The default screen size is adjusted to the resolution of the device.

You change the screen resolution in the runtime settings of the device.

Configuring a fixed screen resolution

To change the screen resolution, follow these steps:

1. Open the runtime settings of the device.
2. Select the screen resolution under "General > Screen > Screen resolution".



The screen resolution is adapted.

Scrolling in the screen

If not all objects are visible in the screen, a scroll bar appears on the right edge of the screen.

To view all objects, move the scroll bar up and down with the mouse.

	Tip for working effectively
• Place the mouse cursor in the screen. Move forward or backward with the mouse wheel to scroll up and down in the screen.	

4.1.3 Using styles

4.1.3.1 Basics on working with styles

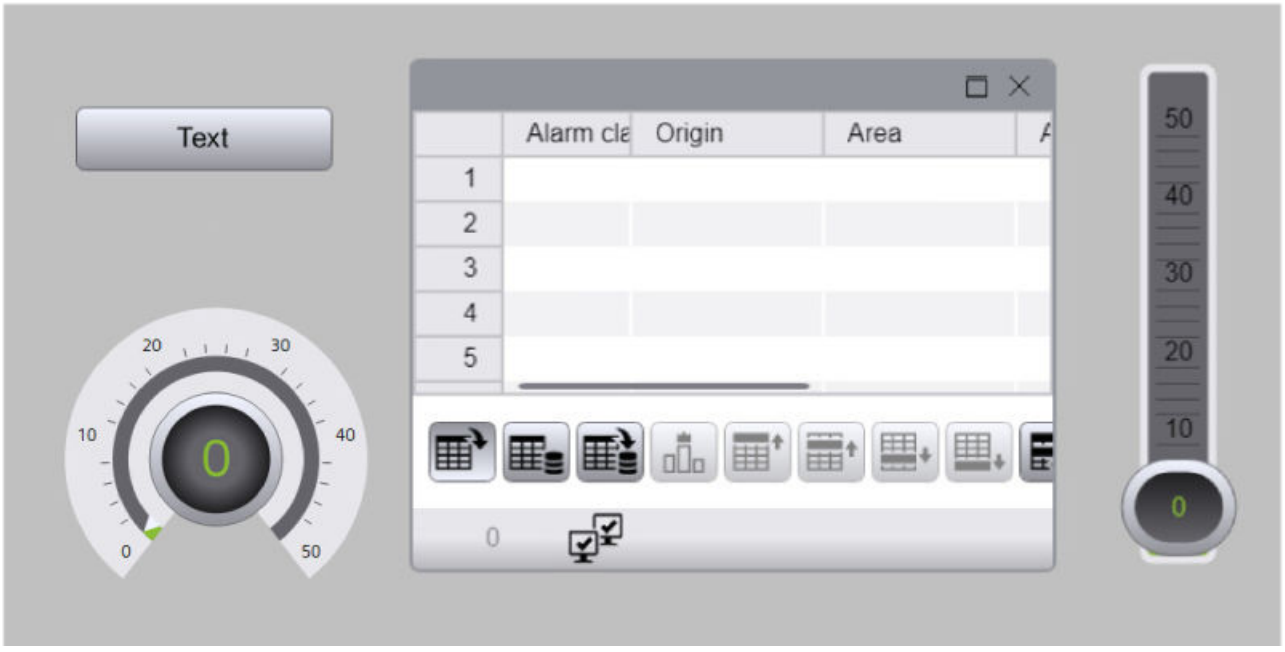
You can assign a predefined style to the objects of the WinCC Unified device.

You can choose from the bright, dark or extended style.

Bright style

Dark style

Expanded style



4.1.3.2 Defining the style

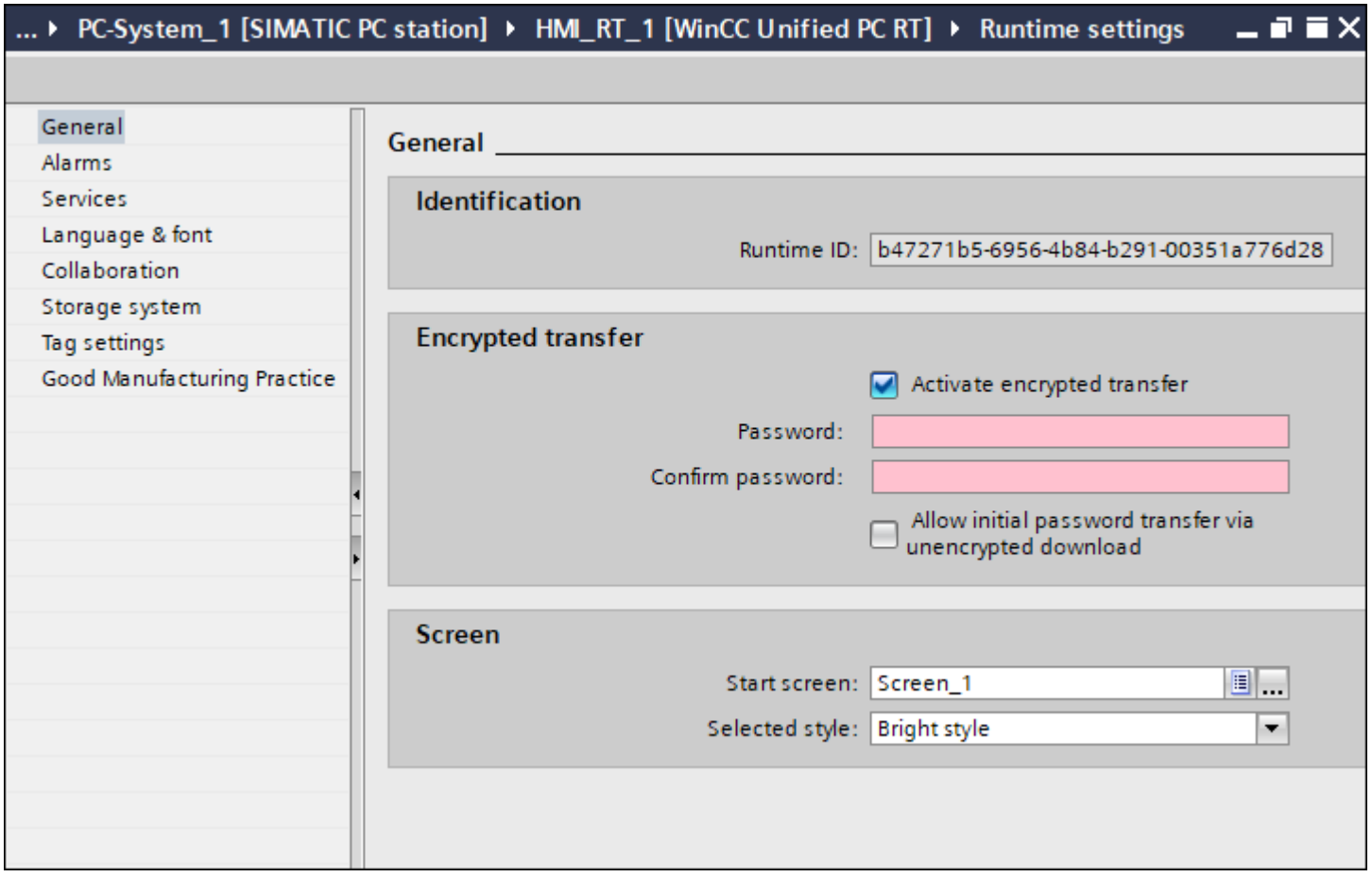
Introduction

You can specify predefined styles for Unified devices. Predefined styles are write-protected and cannot be changed.

Defining the style

To specify a style, follow these steps:

1. Open the "Runtime settings" folder in the project tree.
2. Select the desired style under "General > Screen":
   - Bright style
   - Dark style
   - Expanded style



Result

You have specified a new style that is applied to objects or projects.

4.1.3.3 Switching styles by means of user-defined functions

In runtime, you can change the style with user-defined functions.

```
//Switch to bright style
HMIRuntime.UI.Style = "FlatStyle_Bright";
//Switch to dark style
HMIRuntime.UI.Style = "FlatStyle_Dark";
//Switch to extended style
HMIRuntime.UI.Style = "ExtendedStyle";
```

Determining the currently defined style in runtime

In runtime, you can determine the style with user-defined functions.

```
let MyStyle;
MyStyle = HMIRuntime.UI.Style;
HMIRuntime.Trace("My current style is: " + MyStyle);
```

4.1.4 Task cards

Introduction

The following task cards are available in the "Screens" editor:

- Toolbox: Display and operating objects
- Layout: Aid for customizing the display
- Tasks: "Find and replace" function and language selection
- Libraries: Administration of the project library and of the global libraries

Toolbox

The "Toolbox" task card contains objects in different palettes:

- Basic objects
- Elements
- Controls
- My controls
- Graphics
- Dynamic widgets

You paste objects from the palettes into the screens by drag&drop or a double click. The objects available for selection are determined by the features of the HMI device you are configuring.

You can toggle between the following views in the "Toolbox" task card:

- Thumbnails view 
- List view 

In the symbol view you can switch the labeling of the objects on or off in the shortcut menu.

Layout

The "Layout" task card contains the following palettes for displaying objects and elements:

- Layers: Serves to manage screen object layers The layers are displayed in a tree view and contain information about the active layer and the visibility of all layers.
- Objects out of range: Objects that lie outside the visible area are displayed with name, position and type.

Tasks

The "Tasks" task card contains the following palettes:

- Find and replace: Used to search within an open editor. It includes all options that you need for an efficient search. You have the option of replacing hits individually or automatically replacing all the found texts.
- Languages and resources: Used to select the editing and reference language.

Libraries

The "Libraries" task card show the following libraries in separate palettes:

- Project library: In the "Project library" palette, you can store the library elements that you want to use more than once in the project. The project library is stored together with the project.
- Global library: In the "Global libraries" palette, you manage the global libraries whose library elements you want to reuse over several projects. The global library is stored in a separate file in the specified path on your configuration PC.
- Info (project library): The following is displayed in the "Info" palette:
  - The contents of the library elements
  - The individual versions of types
  - The last date of change of the version

4.1.5 Defining the start screen:

Introduction

The start screen is the initial screen displayed when the project is started in runtime. Define a start screen for each target system. From the start screen, the operator navigates to the other screens.

To compile and download a project, a screen must be defined as the start screen in the project.

Requirement

- A WinCC Unified PC or a Unified Comfort Panel is installed.
- At least one HMI screen is open.

Defining the start screen

To define the start screen, follow these steps:

1. In the project tree, right-click on the screen that you want to define as the start screen.
2. Select "Define as start screen" in the shortcut menu.

Alternatively, right-click in the open screen editor. Select "Define as start screen" in the shortcut menu.

The selected screen is displayed as the start screen when the project is started in runtime.

4.1.6 Screen zooming

Introduction

You can zoom the screen from the zero point. The zero point is located in the upper left corner of the screen.

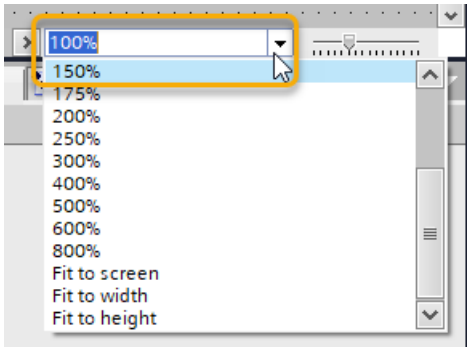
Requirement

- A screen is open.

Zooming the screen from the zero point

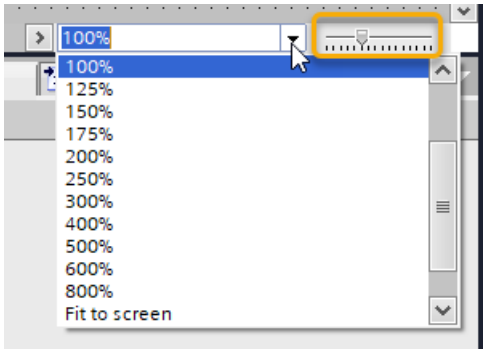
To zoom a screen from the zero point, select one of the following options:

- Click the selection list in the lower-right corner of the screen editor.



  - Select a percentage value.
    The screen is enlarged or reduced by the selected percentage value.
  - Select "Fit to screen", "Fit to width" or "Fit to height".
    The screen is adapted to the window size, the width or the height of the screen.

- Click the scroll bar to zoom in or out of the screen.



  The screen is enlarged or reduced by 25% with each click.

- Press <Ctrl>. Move the mouse wheel forward or backward at the same time to enlarge or reduce the screen.

Objects in the zoomed screen

The size of the objects in the screen is adapted proportionally to the zoom factor.

For objects in a screen with a small zoom factor, the bounding box and the handles are adjusted proportionally to the zoom factor.

4.2 Overview of screen objects

4.2.1 Show object type and name in the tooltip

Introduction

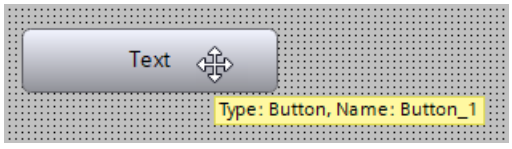
You can display the type and name of the object in the screen.

Displaying the object type and object name in the tooltip

To show the type and name of the object in the tooltip, follow these steps:

1. Move the cursor over the object in the screen.
2. Hover with the cursor over the object.

The type and name of the object are displayed in a tooltip.

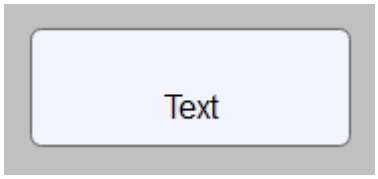


4.2.2 Basic objects

4.2.2.1 Text box

Use

The "Text box" is a closed object which you can fill with a color. You create a text in the text box.



Layout

In the Inspector window, you customize the position, geometry, style, color, and font types of the object. You can adjust the following properties in particular:

- "Text": Specifies the text for the text box.
- "Text trimming": Specifies whether ellipsis is to be displayed after a line break for a long text.
- "Text break": Specifies whether the next word is to be automatically moved to the next row for a long text.

Text

To define a text for the text box, follow these steps:

1. In the Inspector window, click "Properties > Properties > General > Font".
2. Select a font.
3. In the Inspector window, click "Properties > Properties > General > Text" .
4. Enter a text.

	**Efficiency tip**
• Separate the words in the input window "Text" with <Shift+Return>. The words are displayed in individual rows in the text box.	

Direct text input

To change the labeling in the text box directly via the keyboard, follow these steps:

1. Select the text box.
2. Double-click in the text box and enter a text.

Note the following special information:

- Diacritics, such as ä ê ñ, can only be entered if the keyboard layout provides a key for this character. Key sequences such as <`a> for à are not recognized.
- It is not possible to enter Unicode characters using Alt codes.
- Asian language characters cannot be entered using an Input Method Editor (IME).

If you need such characters in the label, you have the following options:

- Use a keyboard layout on which this character is present as a key.
- Copy the character or full label from any source and paste it into the selected object.
- Edit the label in the Inspector window under "Properties > Properties > General > Text".

Trimming text

You can specify ellipsis characters for a text that cannot be displayed in full in the text box.

1. In the Inspector window, click "Properties > Properties > Format > Text trimming".
2. In the "Static value" column, select the option "With character ellipsis".

The text displayed is truncated with ellipsis.

Enabling line breaks

You can enable line breaks for a text that cannot be displayed in full in the text box. If you find that the text box is large enough for display with line breaks:

1. In the Inspector window, click "Properties > Properties > Format > Text break".
2. Select the option "Word wrap" in the "Static value" column.

The text is displayed in full with line breaks.

Dynamizing a text box with text list

To dynamize the text box with text list, follow these steps:

1. Create a text box.
2. Select a resource list in the "Dynamization" column in the Inspector window under "Properties > Properties > General > Text".
3. Select an existing tag or create a new tag using the "Add" button under "Resource list > Settings > Tag".
4. Select an existing text list or add a new text list using the "Add" button under "Resource list > Settings > Resource list".

	**Efficiency tip**
• You can create a text box by dragging-and-dropping a text list from the detail view of the text and graphics lists into the screen. The text box is linked to the text list. Configure the tags whose values determine the display in the text box.	

Dynamizing a text box

Drag-and-drop a tag from the detail view of the tag table directly into the text box. The text box is linked to the tag and the tag name appears in the text box.

See also

Entering multiline text (Page 407)

Enter text directly into the object (Page 406)

4.2.2.2 Graphic view

Use

The "Graphic view" object is used to display graphics.

Use the following graphic formats in the "Graphic view" object: *.bmp, *.ico, *.gif, *.tiff, *.png, *.svg, *.jpeg, *.jpg. The graphic format *.gif is displayed animated. You can also use graphics as OLE objects in the Graphic view.

High-resolution graphic objects require a lot of memory in the project and cause long loading times. In addition, the performance in runtime decreases. Use graphic objects with a resolution that is sufficient for a high-quality display in the runtime project. Note the display resolution of the target device and the size in which the graphic object is displayed on the display of the target device. Adapt the resolution of large graphic objects accordingly before using them in your project.

Layout

In the Inspector window, you customize the settings for the position, shape, style, and color of the object. You can adapt the following properties in particular:

- "Graphic": Specifies the graphic file that is displayed in the object.
- "Background graphic - scale": Specifies how the graphic is scaled.

Inserting graphics

1. In the Inspector window, click "Properties > Properties > General > Graphic".
2. In the "Static value" column, click the arrow in the text box. Graphics from the graphics collection are displayed in the preview. You have the following options for inserting a graphic:
   - Select a graphic from the graphics collection.
   - Insert a graphic from a file using the  button.
   - Create a new graphic from an OLE object using the  button.
3. Click "Apply" to insert the graphic in the Graphic view.

	Tips for an efficient procedure
• Drag-and-drop a graphic from the detail view into the screen. A graphic view is created and linked to the graphic.	

Scaling a background graphic

The following modes for scaling graphics are available:

- None
  The graphic is inserted centered into the graphic view. If the graphic is larger than the graphic view, the graphic is displayed incompletely.
- Fill
  The graphic fills the graphic view. This mode can lead to a distortion of the graphic.
- Uniform
  The graphic is fully displayed and without distortion in the graphic view.
- Stretch to fit
  The graphic is adjusted to the size of the graphic view without distortion. As a result, the graphic may not be displayed completely.
- Tiled
  The graphic is displayed in original size, multi-tiled until the graphic view is filled.

To select a mode for scaling the graphic, proceed as follows:

1. Click "Format > Scale background graphic" in the Inspector window.
2. Select the desired mode in the "Static value" column.

Dynamizing a graphic view with graphic list

To dynamize a graphic view with graphic list, follow these steps:

1. Create a graphic view.
2. Select a resource list in the "Dynamization" column in the Inspector window under "Properties > Properties > General > Graphic".
3. Select an existing tag or create a new tag using the "Add" button under "Resource list > Settings > Tag".
4. Select an existing graphic list or add a new graphic list using the "Add" button under "Resource list > Settings > Resource list".

	Tips for an efficient procedure
• You create a graphic view by dragging and dropping a graphic list from the detailed view of the text and graphic lists into the screen. The graphic view is linked to the graphics list. Configure the tags whose values determine the display in the graphic view.	

4.2.2.3 Line

Application

The "Line" object is an open object.

Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Line - Type"
- "Line - Start" and "Line - End"

Dash type

The layout of the line is specified under "Properties > Properties > Appearance > Line - Type" in the Inspector window. The line is shown without interruption if you select "Solid", for example.

Note

The available line types depend on the selected HMI device.

Line start and end

The start and end points of the line are specified under "Properties > Properties > Appearance > Line - start / Line - end" in the Inspector window.

Use arrow points, for example, as the start and end points of the line. The available start and end points depend on the device.

See also

Changing the object size (Page 381)

4.2.2.4 Rectangle

Application

The "Rectangle" is a closed object which you can fill with a color.

Layout

In the Inspector window you can customize the settings for the position, geometry and color of the object. You can adapt the following properties in particular:

- "Corner > Radius": Specifies the horizontal and vertical distance between the corner of the rectangle and the start point of a rounded corner.

Specifying the corners

The corners of the "Rectangle" object can be rounded to suit your requirements. If the "Radius" property for all four corners is set to 0, a standard rectangle without rounded corners is displayed.

To define the layout of the corners, follow these steps:

1. In the Inspector window, click "Properties > Properties > Appearance > Corners".
2. Enter the radius for each corner.

4.2.2.5 Circle

Application

The "Circle" object is a closed object which can be filled with a color or pattern.

Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "Radius": Specifies the size of the circle.

Radius

The radius of the "Circle" object is specified in the Inspector window. The value is entered in pixels.

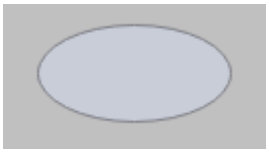
To specify the radius, follow these steps:

1. In the Inspector window, click "Properties > Size and position > Radius".
2. Enter a value of between 0 and 2500.

4.2.2.6 Ellipse

Application

The "Ellipse" is an enclosed object that can be filled with a color or pattern.



Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "Radius X": Specifies the vertical radius of the elliptical object.
- "Radius Y": Specifies the horizontal radius of the elliptical object.

Radius X

The horizontal radius of the "Ellipse" object is specified in the Inspector window. The value is entered in pixels.

1. Click "Properties > Size and position" in the Inspector window.
2. For "Radius X", enter a value between 0 and 2500.

Radius Y

The vertical radius of the "Ellipse" object is specified in the Inspector window. The value is entered in pixels.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter a value between 0 and 2500 for "Radius Y".

4.2.2.7 Polyline

Use

The "Polyline" is an open object. Use the "Polygon" object if you want to fill the object with color.

Layout

In the Inspector window, you customize the settings for the position, shape, style, and color of the object. You can adapt the following properties in particular:

- "Line start" and "Line end": Specifies the type of line start and line end.
- "Points": Modifies, deletes or adds corners.

Line start and end

Define the start and end of the line in the "Properties" Inspector window. Use arrow point, for example, as start and end point. The available start and end points depend on the device.

Points

The corner points are numbered in the order of their creation. You can change, delete, or add more corner points:

1. In the Inspector window, click "Properties > Size and position > Points".
2. Select the required corner point. Enter a value for "X coordinate" and "Y coordinate".
3. Click on the selection button in the "Static value" column to add or delete a corner point. A dialog opens.
4. Use the "Add" command to create a new point. You can delete corner points by selecting the corresponding row in the dialog and selecting "Delete" from the shortcut menu for the row.

Automatic creation of points

To create the points automatically, follow these steps:

1. Select a cell in the "X coordinate" or "Y coordinate" column.
2. Drag the blue border up or down. The value is applied to the target cells.

If you select multiple cells and there is a logical relationship between the values, the values of the destination cells are adapted according to the logical relationship.

Configuring rotation in runtime

You configure the "Polyline" object so that it rotates about a reference point in runtime.

Enter the values for rotation in degrees:

1. In the Inspector window, click "Properties > Size and position".
2. Enter the required values for the following attributes:
   - Pivot point
   - Rotation
   - X pivot point
   - Y pivot point

See also

Changing the object size (Page 381)

Automatically filling in of property values for an object collection (Page 379)

4.2.2.8 Polygon

Use

The "Polygon" is a closed object which you can fill with a background color.

Layout

In the Inspector window, you customize the settings for the position, shape, style, and colors of the object. In particular, you can customize the following property:

- "Points": Modifies, deletes or adds corners.

Points

The corner points are numbered in the order of their creation. You can change, delete, or add more corner points:

1. In the Inspector window, click "Properties > Size and position > Points".
2. Select the required corner point. Enter a value for "X coordinate" and "Y coordinate".

3. Click on the selection button in the "Static value" column to add or delete a corner point. A dialog opens.
4. Use the "Add" command to create a new point.
   You can delete corner points by selecting the corresponding row in the dialog and selecting "Delete" from the shortcut menu for the row.

Automatic creation of points

To automatically create the points for polygons and polylines, follow these steps:

1. Select a cell in the "X coordinate" or "Y coordinate" column.
2. Drag the blue border up or down. The value is applied to the target cells.

If you select multiple cells and there is a logical relationship between the values, the values of the destination cells are adapted according to the logical relationship.

Configuring rotation in runtime

You configure the "Polygon" object so that it rotates about a reference point in runtime.

Enter the values for rotation in degrees.

1. In the Inspection window, click "Properties > Size and position".
2. Enter values for the following attributes in the "Rotation" area.
   - Pivot point
   - Rotation
   - X pivot point
   - Y pivot point

See also

Changing the object size (Page 381)

Automatically filling in of property values for an object collection (Page 379)

4.2.2.9 Circular arc

Use

The "Circular arc" is an open object. Use the "Circle segment" object if you want to fill the object with color. By default, a circular arc is a quarter circle. It can be customized as required.

Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Radius": Define the size of the circular arc.
- "Angle - start" and "Angle - range": Specify where the start and end angle lie on a virtual circle of 360°.

Defining the radius

You define the radius of the "Circular arc" object in the Inspector window. Enter the value in pixels.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter a value for "Radius".

Defining the start angle and angle range

You set the length of the circular arc using the "Angle - start" and "Angle - range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter one value each for "Angle - start" and "Angle - range".

4.2.2.10 Elliptical arc

Use

The "Elliptical arc" is an open object. Use the "Ellipse segment" object if you want to fill the object with color. By default, an elliptical arc is a quarter ellipse. It can be customized as required.

Layout

In the Inspector window you can customize the settings for the object position, geometry, style, frame and color. You can adapt the following properties in particular:

- "Radius": Specifies the size of the elliptical arc.
- "Radius X" and "Radius Y": Specifies the horizontal and vertical radius of the elliptical object.
- "Angle - start" and "Angle - range": Specify where the start and end point lie on a virtual circle of 360°.

Defining the radius

Define the horizontal and vertical radius of the "Elliptical arc" object in the Inspector window. Enter the values using Pixels as the unit.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter one value each for "Radius X" and "Radius Y".

Defining the start angle and angle range

Set the length of the elliptical arc using the "Angle - start" and "Angle - range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties" in the Inspector window.
2. Enter one value each for "Angle - start" and "Angle - range".

4.2.2.11 Circle segment

Use

The "Circle segment" is a closed object that you can fill with a color or pattern. By default, a circle segment is a quarter circle. It can be customized as required.

Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Radius": Define the size of the circle segment.
- "Angle - start" and "Angle - range": Specify where the start and end angle lie on a virtual circle of 360°.

Radius

You define the radius of the "Circle segment" object in the Inspector window. Enter the value using Pixels as the unit.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter a value for "Radius".

Defining the start angle and angle range

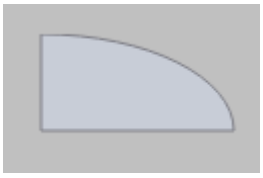
Set the size of the circle segment using the "Angle - start" and "Angle - range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter one value each for "Angle - start" and "Angle - range".

4.2.2.12 Ellipse segment

Use

The "Ellipse segment" is a closed object that you can fill with a color or pattern. By default, an ellipse segment is a quarter ellipse. It can be customized as required.



Layout

In the Inspector window, you customize the settings for the object position, shape, style, and color. You can adapt the following properties in particular:

- "Radius X" and "Radius Y": Specifies the horizontal and vertical radius of the elliptical object.
- "Angle - start" and "Angle - range": Specify where the start and end point lie on a virtual circle of 360°.

Defining the radius

Define the horizontal and vertical radius of the "Ellipse segment" object in the Inspector window. Enter the values using Pixels as the unit:

1. Click "Properties > Size and position" in the Inspector window.
2. Enter one value each for "Radius X" and "Radius Y".

Defining the start angle and angle range

Set the size of the ellipse segment using the "Angle - start" and "Angle - range" attributes. Enter the values using Degrees as the unit.

1. Click "Properties > Size and position" in the Inspector window.
2. Enter one value each for "Angle - start" and "Angle - range".

4.2.2.13 Example: Configuring a rectangle

Task

In this example, you learn how to configure the rectangle.

You configure:

- Color = red
- Black border 2 pixels wide
- Position = (20, 20)
- Size = (100,100)

Changing the color of the rectangle

Follow these steps to change the color of the rectangle:

1. Select the rectangle.
2. Define the background color under "Properties > Appearance > Background - color" in the Inspector window.
3. Select the "Solid" option under "Background - fill pattern".
4. Define the border color under "Properties > Appearance > Border - color".
5. Enter the value "2" for "Border width".

The rectangle is red and has a black border with a width of two pixels.

Repositioning and resizing the rectangle

Follow these steps to change the position and size of the rectangle:

1. Select the rectangle.
2. Enter the value "20" in each case under "Properties > Size and position > Position - left / Position - top".
3. Enter the value "100" in each case under "Properties > Size and position > Size - width / Size - height".

Result

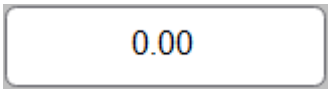
The rectangle is positioned at the coordinates (20, 20), and has a width and height of 100 pixels.

4.2.3 Elements

4.2.3.1 IO field

Use

The "IO field" object is used to enter and display process values.



	Tips for working effectively
• You can create an IO field by moving a configured tag from the detail view onto the screen using drag-and-drop. An IO field is created and linked to the tag. • If you select multiple tags and move them from the detail view onto the screen using drag-and-drop, an IO field is created for each tag which is linked to the respective tag.	

Layout

In the Inspector window, you customize the settings for the position, shape, style, color and font types of the object. You can adapt the following properties in particular:

- "Mode": Specifies whether the values are entered and displayed in Runtime or if the values are only displayed.
- "Reaction to input": Specifies the behavior of the object in Runtime.
- "Hidden input": Specifies whether the input value is displayed normally or encrypted during input.

Mode

You can define the behavior of the IO field in the Inspector window under "Properties > General > Mode".

Mode	Description
"Input/output"	Values can be input and output in the IO field in runtime.
"Output"	The IO field is used for the output of values only.

Note

Reports

In reports, IO fields only output data. "Output" mode is preset. Properties for configuring input are not available, for example "Hidden input".

Connection to Char data type

If you connect the IO field to a controller tag of data type "Char", the following restrictions apply:

- The input accepts digits only: 0 ... 9.
  The entered string of digits is converted to the corresponding character according to the ASCII table.
  Example: Input 6 + 5 becomes "A".
- Only enter digit sequences between 0 and 129.

To input alphanumeric characters directly, use the alternative data type "WChar", for example.

Hidden input

In runtime, the input can be displayed normally or encrypted, for example for hidden input of a password. A "*" is displayed for each character in hidden input. The data format of the value entered cannot be recognized.

1. In the Inspector window, select "Properties > Miscellaneous > Reaction to input".
2. Select "Hidden input".

Value range for the LTIME data type

LTime values are saved as 64-bit Int with sign. For HMI tags with LTime data type:

Value range	-9223372036854775808 to 9223372036854775807
Unit	100 ns

Note

Setting an LTime PLC tag via HMI

S7-1500 tags with data type LTime have the unit nanoseconds (ns). HMI user inputs in IO fields that are linked with such tags are converted to ns when the value is sent to the controller.

Note

MAX_SAFE_INTEGER

Depending on the Javascript engine of the web client, the actual value may lose accuracy during communication between the HMI device and the controller due to rounding if it is outside the value range of MAX_SAFE_INTEGER.

Additional information on MAX_SAFE_INTEGER can be found here (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER).

4.2.3.2 Symbolic IO field

Use

You can use the "Symbolic IO field" object to configure a selection list for input and output of texts or graphics in runtime.

Note

Selecting the default entry is not possible in runtime.

Layout

In the Inspector window, you customize the settings for the position, shape, style, color and font types of the object. You can adapt the following properties in particular:

- "Mode": Specifies the response of the object in runtime.
- "Resource list": Specifies the text or graphic list that will be linked with the object.

Mode

The response of the symbolic IO field is specified in the Inspector window in "Properties > Properties > Miscellaneous > Mode".

Mode	Description
"Output"	The symbolic IO field is used for the output of values.
"Input/output"	The symbolic IO field is used for the input and output of values.

Linking symbolic IO field with text list

To link a text list with the symbolic IO field, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Resource list".
2. Open the selection list for "Text list" in the "Static value" column.
3. Select a text list.
4. Open the "Text and graphic lists" editor in the project tree.
5. Select the "Text lists" tab. Click on the selected text list.
6. Select an entry in the "Text list entries" table as the default entry. The text from the default entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Text" column is displayed in the object.

	Tips for working effectively
•	Drag a text list from the detail view of the text and graphic lists to the symbolic IO field. The symbolic IO field is linked to the text list.

Linking symbolic IO field with graphic list

To link a graphic list with the symbolic IO field, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Resource list".
2. Open the selection list for "Graphic list" in "Resource list".
3. Select a graphic list.
4. Open the "Text and graphic lists" editor in the project tree.
5. Select the "Graphic lists" tab. Click on the selected graphic list.
6. Select an entry in the "Graphic list entries" table as the default entry. The graphic from the standard entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Graphic" column is displayed in the object.

	Tips for working effectively
•	Use drag-and-drop to move a graphic list from the detail view of the text and graphic lists to the symbolic IO field. The symbolic IO field is linked to the graphic list.

Dynamizing a symbolic IO field

To dynamize the symbolic IO field using a tag, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Process value".
2. Select "Tag" in the "Dynamization" column. The "Tag" page will open.
3. Select:
   – Select an existing tag under "Tag > Process > Tag", or
   – Create a new tag using the "Add" button.

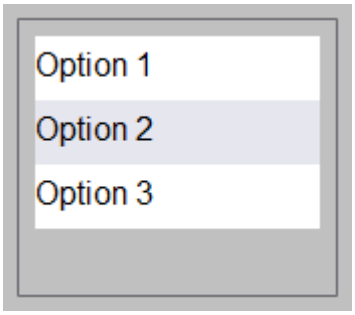
	Tips for working effectively
•	Use drag-and-drop to move a tag from the detail view of the tag table to the symbolic IO field. The symbolic IO field is linked to the tag.

4.2.3.3 List box

Use

You use the "List box" object to present and select multiple list entries. You activate list entries by default so that the operator only changes the preset entry if necessary. If the list box is larger than the bounding box, WinCC automatically adds a scroll bar to the right margin.

To incorporate list boxes into the process, dynamize the corresponding properties.



Layout

In the Inspector window, you customize the position, style, colors and font type settings of the object. You can adjust the following properties in particular:

- "Selection items": Defines the list entries.
- "Select item": Defines which entry is displayed as activated by default.

Defining the number of entries

You specify the number of entries in the Inspector window:

1. In the Inspector window, select "Properties > General > Selection items".
2. Click on the selection button in the "Static value" column.
   A dialog opens.
3. Specify the desired number of entries with "Add".
   To delete entries, click in the corresponding line and press the <DEL> key or click ✕.
4. To change the order of the entries, click in the corresponding line and move the entry using the icons ↑ ↓.

Specifying the default value of the list entry

Use the "Select item" property of a selection item to specify which list item is to be shown as enabled.

You can activate multiple options.

To do so, select the check box in the "Static value" column of the "Select item" property of the respective selection item.

Dynamizing a list box

Drag-and-drop a tag from the detail view of the tag table into the list box. The list box is linked to the tag.

Dynamization of graphic properties with tags or scripts

You can dynamize the following property containing a graphic with a tag or with a script:

- Graphic

See also

Automatically filling in of property values for an object collection (Page 379)

Entering multiline text (Page 407)

4.2.3.4 Button

Use

The "Button" object allows you to configure an object that the operator can use in runtime to execute a configurable function.

Layout

In the Inspector window, you customize the position, geometry, style, color, and font types of the object. You can adjust the following properties in particular:

- "Type": Define the graphic display of the object.
- "Background graphic - scale": Specify how the graphic is scaled.

Defining the content

You can specify how the button is displayed under "Properties > General > Content > Type" in the Inspector window.

Type	Description
"Text"	The button is displayed with text. This text explains the function of the button.
"Graphic"	The button is displayed with a graphic. This graphics represents the function of the button.
"Graphics or text"	The button is displayed with text or graphics. If the graphics cannot be displayed, the corresponding text is displayed.
"Graphics and text"	The button is displayed with text and graphics.

Different options are available depending on the device.

Scaling a background graphic

	Tips for working effectively
•	You add a graphic to the button by dragging-and-dropping the graphic from the detail view of the project graphics to the button.

The following modes for scaling graphics in buttons are available:

- None
  The graphic is inserted centered inserted into the button. If the graphic is larger than the button, the graphic will be displayed incompletely.
- Fill
  The graphic fills the button. This mode can lead to a distortion of the graphic.
- Uniform
  The graphic is fully displayed and without distortion in the button.
- Stretch to fit
  The graphic is adjusted to the size of the button without distortion. This may cause the graphic to be displayed incompletely.

To select a mode for scaling the graphic, follow these steps:

1. In the Inspector window, select "Properties > General > Content > Background graphic - scale".
2. Select the desired mode in the "Static value" column.

Text / Graphic

Depending on the "Content" property, define whether the display is static or dynamic. The display is defined under "Properties > General > Text" or "Graphic" in the Inspector window.

Your options for the type "Text" or "Graphic" include the following.

Type	Description
"Text"	Use "Text with pressed button" to specify the text displayed in the button for the "ON" state.
"Graphic"	Use "Graphic with pressed button" to specify a graphic displayed in the button in the "ON" state.

Direct text input

To change the labeling in the button directly via the keyboard, follow these steps:

1. Selecting the button.
2. Double-click in the button and enter a text.

Note the following special information:

- Diacritics, such as ä ê ñ, can only be entered if the keyboard layout provides a key for this character. Key sequences such as <`a> for à are not recognized.
- It is not possible to enter Unicode characters using Alt codes.
- Asian language characters cannot be entered using an Input Method Editor (IME).

If you need such characters in the label, you have the following options:

- Use a keyboard layout on which this character is present as a key.
- Copy the character or full label from any source and paste it into the selected object.
- Edit the label in the Inspector window under "Properties > Properties > General > Text".

Configuring a screen change

1. Drag a button from the "Toolbox" task card to a screen.
2. Configure the "ChangeScreen" system function to an event of the button.
3. Add another screen in the "Value" column.

	**Tips for working effectively**
• You can configure a screen change by dragging-and-dropping a configured screen from the project tree into the open screen. A button is automatically created and linked to the screen.	

Dynamizing a button with text list

Drag-and-drop a text list from the detail view of the text and graphic lists to the button. The button is linked to the text list.

Dynamizing a button with graphic list

Drag-and-drop a graphic list from the detail view of the text and graphic lists directly to the button. The button is linked to the graphic list.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Graphic - pressed button

See also

Enter text directly into the object (Page 406)

Entering multiline text (Page 407)

4.2.3.5 Switch

Use

The "Switch" object is used to configure a switch that is used to switch between two predefined states in runtime. The current state of the "Switch" object can be visualized with either a label or a graphic.

The following figure shows a "Switch" type switch.

Layout

In the Inspector window, you customize the position, geometry, style, color, and font types of the object. In particular, you can customize the following property:

- "Type": Specifies the graphic representation of the object.

Type of representation

You can specify how the switch is displayed under "Properties > General > Content > Type" in the Inspector window.

Type	Description
"Graphic"	The current state of the switch is shown with a graphic. In runtime click on the button to actuate the switch.
"Text"	The current state of the switch is shown with a label. In runtime click on the button to actuate the switch.
"Graphics or text"	The switch displays graphics or a text. If the graphics are not available, the text is displayed.
"Graphics and text"	The switch displays graphics and a text.

Dynamizing a switch with text list

Drag-and-drop a text list from the detail view of the text and graphics lists directly to the switch. The switch is linked to the text list.

Dynamizing a switch with graphics list

Drag-and-drop a graphics list from the detail view of the text and graphics lists to the switch. The switch is linked to the graphics list.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Graphic - pressed button

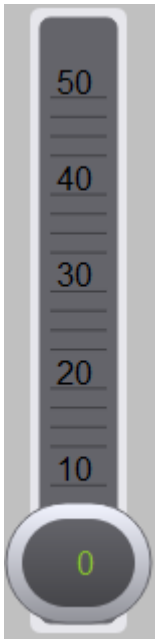
See also

Entering multiline text (Page 407)

4.2.3.6 Bar

Use

The tags are displayed graphically using the "Bar" object. The bar graph can be labeled with a scale of values.



Layout

In the Inspector window, you customize the settings for the position, shape, style, color, and font types of the object. You can adapt the following properties in particular:

- "Trend indicator - show": Shows whether the current value is higher or lower than the previous value.
- "Process value indicator - mode": Specifies how the process value is displayed in the bar chart.
- "Scale": Specifies the properties for the bar scale.

When the object in the light or dark style does not meet the following dimensions in Runtime, it is automatically displayed in compact mode:

- Vertical alignment: 100 pixels high or 30 pixels wide
- Horizontal alignment: 30 pixels high or 100 pixels wide

Displaying the process value indicator

You can use the "Process value indicator - mode" property to select the process value of the selected tags in the bar in runtime:

1. In the Inspector window, click "Properties > Miscellaneous > Process value indicator - mode".
2. Select another "Indicator" mode in the "Static value" column.
3. Go to "Process value indicator - foreground color" and select the display color for the process value.

Define bar segments

You can define the settings for the bar scale under "Properties > General > Scale":

- "Scaling type": Specifies how the bar scale is calculated, for example "Linear".
- "Alignment": Specifies whether the bar is displayed horizontally or vertically.
- "Scale mode": Specifies whether the scale is subdivided with tick marks, numbers, or not at all.
- "Scale value - maximum" and "Scale value - minimum": Specifies the start and end value displayed on the scale.

Defining scale gradation

Use the "Division count" property to define the subdivision count for the bar scale divisions.

The "Subdivision count" property defines the number of ticks between the division marks.

1. In the Inspector window, click "Properties > General > Scale".
2. Enter the required values for the "Division count" and "Subdivision count".

Note

The division count can only be changed if "Automatic scaling" is disabled.

Dynamizing bars

Drag a tag from the detail view of the tag table into the bar. The bar is linked to the tag.

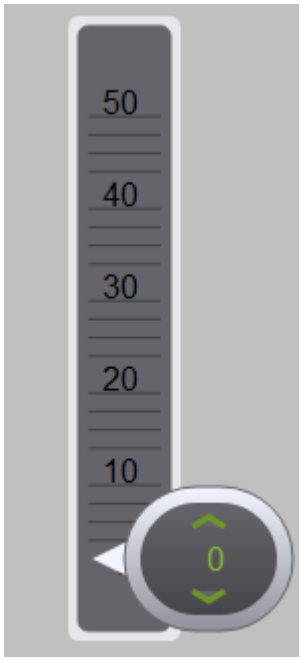
See also

Entering multiline text (Page 407)

4.2.3.7 Slider

Use

Process values are monitored and adapted within a defined range with the "Slider" object. The monitored range is visualized in the form of a slider. By adjusting the slider, you intervene in the process and correct the displayed process value.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can, in particular, adapt the following properties as required:

- "Scale value - maximum" and "Scale value - minimum": Specifies the top and bottom values of the scale.
- "Process value indicator - mode": Specifies how the current process value is displayed in the slider.
- "Trend indicator - show": Specifies how the current value has changed compared to the previous values.

When the object in the light or dark style does not meet the following dimensions in Runtime, it is automatically displayed in compact mode:

- Vertical alignment: 100 pixels high or 30 pixels wide
- Horizontal alignment: 30 pixels high or 100 pixels wide

Maximum and minimum scale value

The top and bottom end values of the scale are specified in the Inspector window.

1. In the Inspector window, click "Properties > Properties > General > Scale".
2. Enter a number each at "Scale value - maximum" and "Scale value - minimum". If you select a tag as the end value of the scale, the number will be no longer available.

Show value

Specify that the value of the current position is displayed below the slider in the Inspector window.

1. Click "Properties > Miscellaneous" in the Inspector window.
2. Select "Value - show".

Process value indicator - mode

Specify a mode for process value display:

1. In the Inspector window, click "Properties > Miscellaneous > Process value indicator - mode".
2. Select a mode in the "Static value" column.

Mode	Description
Bar	Displays the bar with the process value indicator.
Indicator	Shows the process value indicator as a position on the bar.
Detailed indicator	Shows the process indicator in the bar.
Bar with detailed indicator	Shows the current process value and its position in the slider bar.

Note

If you have configured a tag for the "Process value indicator mode" property of a slider, changing the tag in runtime has no effect on the slider.

Dynamizing a slider

Drag a tag from the detail view of the tag table into the slider. The slider is linked to the tag.

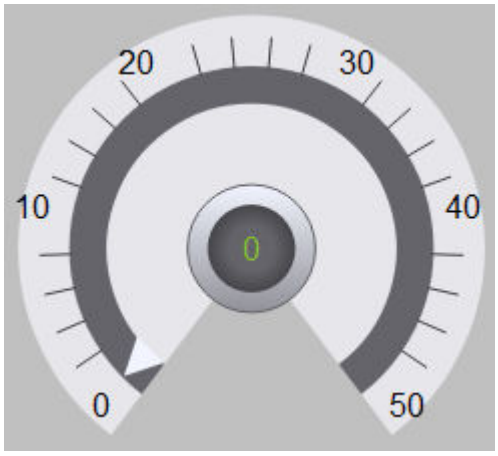
See also

Entering multiline text (Page 407)

4.2.3.8 Gauge

Use

The "Gauge" object shows numeric values in the form of an analog gauge. For example, a glance in runtime is enough to note that the boiler pressure is in the normal range. The gauge is for display only and cannot be controlled by the operator.



Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Peak indicator": Specifies whether the measurement range is indicated with a peak indicator.
- "Scale value - maximum" and "Scale value - minimum": Specifies the top and bottom values of the scale.
- "Normal range - color": Specifies the color in which the normal range is displayed.
- "Scale": Specifies various settings for the scale view.

When the object in the light or dark style is less than 180 pixels tall or wide in Runtime, it is automatically displayed in compact mode.

Display peak value

The "Peak indicator" property can be used to enable a marker function for the maximum or minimum pointer movement in runtime.

1. Click "Properties > Miscellaneous > Peak indicator" in the Inspector window.
2. Select the option "High" or "Low" in the "Static value" column.

Scale value - maximum and scale value - minimum

You can set the top and bottom end values of the scale in the Inspector window.

1. In the Inspector window, click "Properties > General > Scale".
2. Enter a number each at "Scale value - maximum" and "Scale value - minimum".
   If you select a tag as the end value of the scale, the number will be no longer available.

Configuring a scale

1. In the Inspector window, click "Properties > General > Scale".
2. Under "Angle - start", specify the angle at which the scale is to start. The angle is specified in degrees, starting at the zero position.
   The scale runs clockwise. A starting value of 0 corresponds to a display of 3 o'clock.
3. Under "Angle - range", specify the range in degrees to be covered by the scale.
4. Under "Scale mode", specify whether the divisions are displayed as ticks or numbers.

Dynamizing a gauge

Drag a tag from the detail view of the tag table into the gauge. The pointer instrument is linked to the tag.

4.2.3.9 Clock

Use

The "Clock" object displays the time.

Layout

In the Inspector window, you customize the position, geometry, style, color and font types of the object. You can adapt the following properties in particular:

- "Clock face - mode": Specifies whether the hour marks of the analog clock are displayed as ticks or numbers.
- "Hand - show hours", "Hand - show minutes" and "Hand - show seconds": Specifies whether the hour hand, minute hand and second hand are displayed on the clock.

When the object in the light or dark style is less than 100 pixels tall or wide in Runtime, it is automatically displayed in compact mode.

Configuring the clock face

In the Inspector window, you can specify how the hour marks are displayed.

1. In the Inspector window, click "Properties > General > Clock face - mode".
2. Select "Ticks" to display hours as ticks.
   Alternatively, select "Numbers" for a numerical display of the hours in the view.

Dynamizing the clock

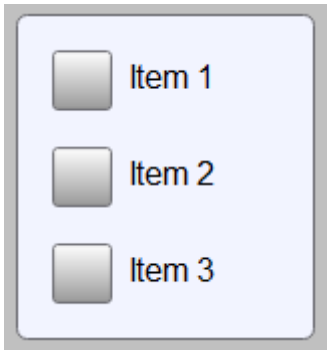
Drag-and-drop one tag from the detail view of tag table into the clock. The clock is linked to the tag.

4.2.3.10 Check box

Use

You use the "Check box" object to display and select multiple entries. You activate a selection item by default so that the operator only changes the preset value if necessary. The operator can select multiple options in runtime. You can specify a text or a graphic for each option.

To integrate the check box into the process, dynamize the corresponding properties.



Layout

In the Inspector window, you customize the position, geometry, style, color, and font types of the object. You can adjust the following properties in particular:

- "Selection items": Defines the number of options.
- "Select item": Defines which entries are displayed as activated by default.

Note

The item height option of the radio button is set to "0" during the creation of a new object. This value does not represent the actual value 0, but a default setting.

Defining the number of entries

You specify the number of entries in the Inspector window:

1. In the Inspector window, select "Properties > General > Selection items".
2. Click on the selection button in the "Static value" column.
   A dialog opens.
3. Specify the desired number of entries with "Add".
   To delete entries, click in the corresponding line and press the <DEL> key or click ✕.
4. To change the order of the entries, click in the corresponding line and move the entry using the icons ↑ ↓.

Using graphics and texts in the selection items

You can mark the selection items with texts or graphics. The following modes are available:

- "Graphic and text": The selection item shows text and graphic.
- "Graphic or text" The selection item is visualized either by a graphic or a text. If the graphic is not available, the text is displayed.
- "Graphic": The selection item is visualized with a graphic.
- "Text": The selection item is visualized with an inscription.

To configure the CheckBox contents, follow these steps:

1. Under "Properties > Format > Content > Type" select the type for display of the selection items, e.g. "Graphic and text".
2. Under "General > Selection items > [x] Selection item > Text" enter the text that is to be shown in the check box as the selection item.
3. Under "Selection items > [x] Selection item > Graphic", open the selection list.
4. Select the appropriate graphic.

Specify default of the check box

Use the "Select item" property of a selection item to define whether it is to be shown as enabled in a check box list.

You can activate multiple options.

To do so, select the check box in the "Static value" column of the "Select item" property of the respective selection item.

Dynamizing a check box

Drag-and-drop a tag from the detail view of the tag table directly to the check box. The check box is linked to the tag.

Dynamization of graphic properties with tags or scripts

You can dynamize the following property containing a graphic with a tag or with a script:

- Graphic

See also

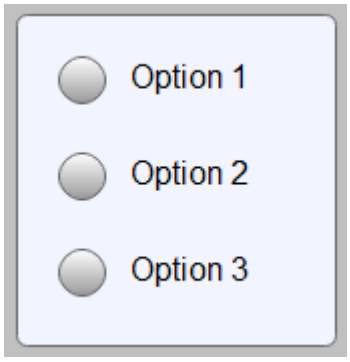
Automatically filling in of property values for an object collection (Page 379)

Entering multiline text (Page 407)

4.2.3.11 Radio button

Use

You can use the "Radio button" object to display and select various options. Only one of these options can be selected by the operator. Enable one of the options by default so that the operator only changes the default value if necessary. To incorporate a radio button into the process, dynamize the corresponding attributes.



Layout

In the Inspector window, you can customize the settings for the position, geometry, style, and color of the object. You can adjust the following properties in particular:

- "Selection items": Defines the number of options.
- "Select item": Defines which entry is displayed as activated by default.

Note

The item height option of the radio button is set to "0" during the creation of a new object. This value does not represent the actual value 0, but a default setting.

Defining the number of entries

You specify the number of entries in the Inspector window:

1. In the Inspector window, select "Properties > General > Selection items".
2. Click on the selection button in the "Static value" column.
   A dialog opens.
3. Specify the desired number of entries with "Add".
   To delete entries, click in the corresponding line and press the <DEL> key or click ✕.
4. To change the order of the entries, click in the corresponding line and move the entry using the icons ↑ ↓.

Using graphics and texts in the selection items

You can mark the selection items with texts or graphics. The following modes are available:

- "Graphic and text": The selection item shows text and graphic.
- "Graphic or text" The selection item is visualized either by a graphic or a text. If the graphic is not available, the text is displayed.
- "Graphic": The selection item is visualized with a graphic.
- "Text": The selection item is visualized with an inscription.

To configure the contents in the radio button, follow these steps:

1. Under "Properties > Content > Type", select the type for display of the selection items, e.g. "Graphic and text".
2. Under "Selection items > [x] Selection item > Text" enter the text that is to be shown in the check box as the selection item.
3. Under "Selection items > [x] Selection item > Graphic", open the selection list.
4. Select the appropriate graphic.

Specifying the default value of the radio button

Use the "Select item" property of a selection item to specify which radio button item is to be shown as enabled.

You can only enable one item.

Under "Properties > Selection items", activate the "Select item" property of the item to be activated by default.

Dynamizing a radio button

Drag-and-drop a tag from the detail view of the tag table into the radio button. The radio button is linked to the tag.

Dynamization of graphic properties with tags or scripts

You can dynamize the following property containing a graphic with a tag or with a script:

- Graphic

See also

Automatically filling in of property values for an object collection (Page 379)

Entering multiline text (Page 407)

4.2.3.12 Touch area

Use

The "Touch area" object allows you to configure an object that the operator can use in runtime to execute any configurable function. A gesture on the user interface starts the execution of the function. The gesture is recognized in the area where it begins.

Layout

In the Inspector window, you customize the settings for the position, shape and color of the object. The "Touch area" object is shown as a dotted area in the engineering system.

Define gestures

The "Touch Area" object distinguishes between the following gestures:

- Right
- Left
- Up
- Down

To distinguish between the gestures, program a J-Script that evaluates the gesture.

1. Click  in the Inspector window under "Properties > Events > Gesture detected".
2. Copy the code example into the programming window.

Code example

```
export function Touch_area_1_OnGestureDetected(item, gesture) {
// value of tag ‚MyTag1' will be set depending on the detected gesture
if(gesture == UI.Enums.HmiGesture.SwipeRight)
{
UI.RootWindow.Screen = 'ScreenRight';
let tag1 = Tags('tag1');
tag1.Write(1); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.SwipeLeft)
{
UI.RootWindow.Screen = 'ScreenLeft';
let tag1 = Tags('tag1');
tag1.Write(2); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.SwipeUp)
{
UI.RootWindow.Screen = 'ScreenUp';
let tag1 = Tags('tag1');
tag1.Write(3); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.SwipeDown)
{
UI.RootWindow.Screen = 'ScreenDown';
let tag1 = Tags('tag1');
tag1.Write(4); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.Unknown)
{
let tag1 = Tags('tag1');
tag1.Write(0); //write value '1234' to tag 'MyTag1'
}
}
```

4.2.3.13 Examples

Example: Configuring an IO field

Task

In this example, you learn how to configure an IO field and connect it to a tag.

You configure:

- Color = blue
- Border color = gray

- Mode = Input/output
- HMI tag = MyTag

Requirement

- A project is open.
- A screen is configured.

Configuring an IO field

To configure an IO field, follow these steps:

1. Open the "Elements" palette in the "Toolbox" task card.
2. Drag the "IO field" object onto the screen.
3. In the Inspector window, navigate to "Properties > Appearance > Background - color".
4. In the "Static value" column, select the blue color.
5. Navigate to "Properties > Appearance > Border - color".
6. In the "Static value" column, select the gray color.
7. Select the "Input/output" mode under "Properties > General > Mode".

Connecting the IO field to a tag

To connect the IO field to a tag, follow these steps:

1. In the Inspector window, click "Properties > General > Process value" in the "Dynamization" column.
2. Select the entry "Tag" from the list.
   The "Tag" dialog opens.
3. Click on the selection button under "Tag > Process > Tag". A dialog opens.
4. Click the "Add" button to add "MyTag" tag. Click "OK".
5. Go to "Properties > Miscellaneous > Reaction to input" and set how the values are to be handled in runtime, for example "Accept value after exit".

Result

The IO field has been configured as specified and connected to the tag. In runtime, you can see the current value of the tag in the IO field and can also input the value for the tag. The value is applied to the tag.

Example: Set values

Task

In this example, you learn how to show the process values in runtime, enter the values, or change the values. You also learn how to visualize the display of the actual speed of a motor and how to regulate it.

You configure:

- Two IO fields for the input and output of the process values.
- Two text boxes for describing the IO fields.
- A slider to display and adjust the values.

Requirement

- A project is open.
- A screen is configured.
- The tags "SetValue" and "ActualValue" have been created as process values for the motor speed.

Configuring IO fields

With two IO fields, you can display the current value of the tags in the screen. You can enter the values for the process or change the values.

To configure the IO fields, follow these steps:

1. Insert the "IO field" object from the "Toolbox" task card into the screen.
2. Specify the height, width, and position for the object under "Properties > Size and position".
3. Under "Properties > General > Mode" specify the "Input/output" mode in the "Static value" column.
4. Click under "Properties > General > Process value". In the drop-down list of the "Dynamization" column, select the entry "Tag".
   The "Tag" dialog opens.
5. Under "Tag" specify the "SetValue" tag whose values you want to display and change in runtime.
6. Define an additional IO field for the "ActualValue" tag in "Output" mode.
7. Configure two text boxes, "Actual value" and "Set value", as the IO fields description.

	**Efficiency tip**
• You can also create a new IO field by moving a configured tag from the detail view onto an HMI screen using drag-and-drop. An IO field is created automatically and connected to the desired tag.	

Configuring a slider

You can use a slider to intervene in the process and change the displayed process value.

Follow these steps to configure the slider:

1. Add the "Slider" object to the screen from the "Toolbox" task card.
2. Specify the desired height, width and position for the object under "Properties > Size and position".
3. Under "Properties > Miscellaneous > Process value indicator - mode", specify the "Detailed indicator" mode in the "Static value" column.
4. Click under "Properties > General > Process value".
5. In the drop-down list of the "Dynamization" column, select the entry "Tag".
   The "Tag" dialog opens on the right in the Inspector window.
6. Under "Tag" specify the "SetValue" tag whose values you want to display and change in runtime.

Result

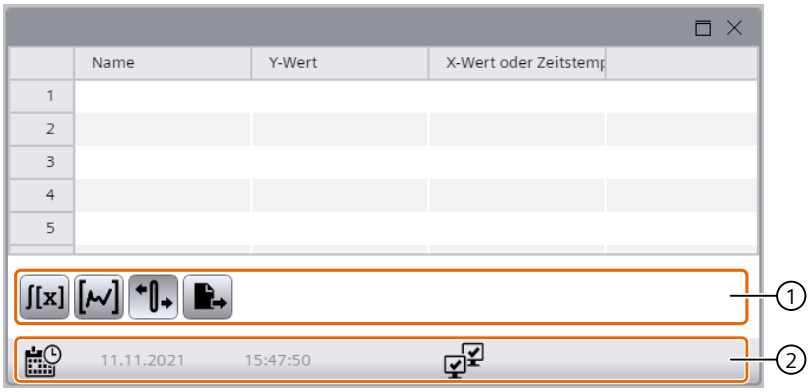
In runtime, the actual motor speed is displayed in the IO field. You can transfer the speed in the IO field "Set value" to the motor. Using the slider, you can read the actual speed and control the speed yourself by moving the slider.

4.2.4 Controls

4.2.4.1 Configuring the toolbar and information bar

Introduction

You can operate the controls in runtime using the buttons in the toolbar. The information bar displays the status messages of the control. During configuration, you define the contents of the toolbar and information bar.



① Toolbar
② Information bar

Requirement

- You have opened the screen which contains at least one object, for example, the trend companion.
- The Inspector window is open.

Configuring the toolbar

To configure the toolbar, follow these steps:

1. In the Inspector window under "Properties > Miscellaneous > Toolbar", configure the general properties of the toolbar, such as background color or visibility.
2. In the Inspector window, under "Properties > Properties > Miscellaneous > Toolbar > Elements > Button > Visibility", enable the buttons that you need in Runtime.

   Note

   Only elements whose visibility is activated in the TIA Portal are transferred to the runtime. Items whose visibility is disabled in the TIA Portal are deleted from the array of elements. You cannot address them in runtime, e.g. via a script.

   If you hide an element, the numbering of the following elements in the runtime changes. Example:
   The parameter set control has 10 elements. Array numbers 0 to 9 are assigned to the elements in the TIA Portal. If you deactivate the visibility of the element with the array number 8, the element with the array number 9 must be addressed in runtime with the number 8.

   If you want to hide an element and still access it, use dynamization, e.g. via a script.
3. Configure the button display, for example, background color, border and size.
4. Under "Properties > Properties > Miscellaneous > Toolbar > Elements > Button > Authorization", select the authorization that is required in Runtime to operate the button.
5. When a button is not operated in Runtime, disable "Allow operator control". You can reactivate a disabled a button by using a script in runtime, for example.

Configuring the information bar

To configure the information bar, follow these steps:

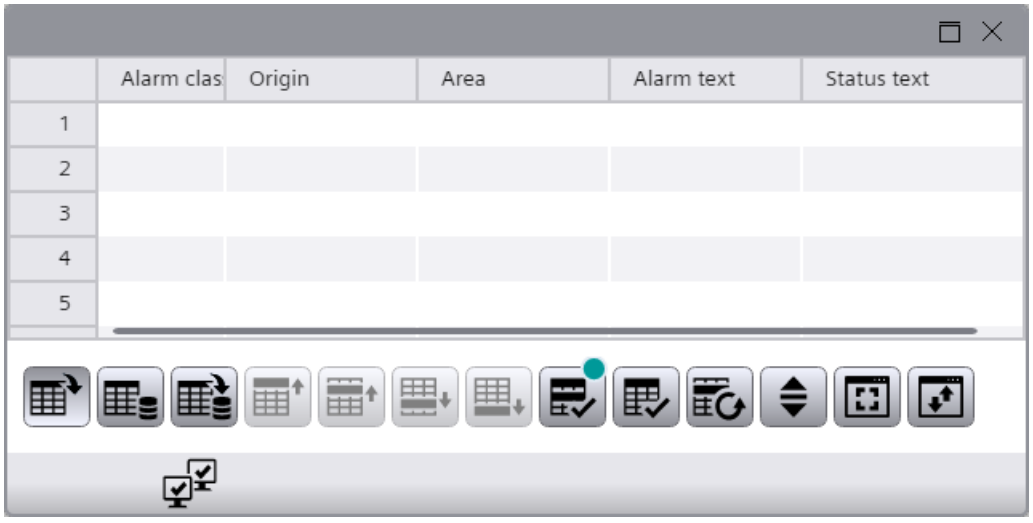
1. In the Inspector window under "Properties > Properties > Miscellaneous > Information bar", configure the general properties of the information bar, such as background color or visibility.
2. In the Inspector window under "Properties > Properties > Miscellaneous > Information bar > Elements > State display > Visibility", enable the elements that you need in Runtime.
3. Configure the display of the respective element.
4. Select the authorization that is required in Runtime to operate the element.
5. When an element is not operated in Runtime, disable "Allow operator control". You can enable a disabled element again, for example, with a script in Runtime.

4.2.4.2 Alarm control

Use

The "Alarm control" object displays alarms that occur during the process in a plant. You can also use the alarm control to display the alarms in lists.

WinCC offers various views, such as "Show active alarms" or "Show logged alarms".



Layout

You can change the settings for the position, geometry, style, color, and font of the object in the Inspector window. You can adapt the following properties in particular:

- "Alarm control": Defines various properties for the display of alarms, e.g. background color and row height.
- "Sorting - allow": Defines whether the alarms are sorted in runtime.
- "Information bar": Defines the elements of the information bar.
- "Toolbar": Defines the buttons of the alarm control.
- "Focus - show visual": Specifies whether the selected properties are visible.

Defining the properties of the alarm control

To define the properties of the alarm control, follow these steps:

1. In the Inspector window, click "Properties > Properties > Miscellaneous > Alarm control".
2. Define settings for the rows and cells, e.g.:
   – "Row height": Defines the height of the rows in the alarm control.
3. Define the settings for the headers under "Header - settings", e.g.:
   – "Row header": Defines whether each row has a header.
   – "Column header": Specifies the representation of the column header.
4. Define the width and color of the grid lines.
5. Define the use of scroll bars.

Configuring output of alarms

Define the following properties to configure the outputs of the alarm control:

- "General > Alarm source": Defines which alarms are displayed in this alarm control.
- "Miscellaneous > Alarms - show current":
  If you activate this property, the following applies in runtime:
  - The most recent alarms are always displayed first in the alarm control.
  - Alarms that have been filtered out of the alarm control are not displayed.
  - The visible area of the alarm control is shifted automatically, as needed.
  - Users cannot select alarms individually or sort them by column.

  If you configure the "Alarms - show current" button as visible and operable, users can pause and start this behavior in runtime as required. The alarm control always starts with the behavior configured via "Miscellaneous > Alarms - show current".
- "Miscellaneous > Alarm statistics settings": Setting options that contribute to the evaluation of the alarm statistics, e.g. start time, maximum number of alarms.

Setting up column sorting

To set up the column sorting, follow these steps:

1. In the Inspector window, click on "Properties > Miscellaneous > Alarm control > Columns > [1] Alarm statistics column".
2. Select the sorting direction and sorting order for the individual columns.

Define the sorting direction of the alarms in the alarm control, e.g. "Ascending", under "Properties > Properties > Format > Starting sorting direction".

Configuring reordering of the columns

Configure whether operators can reorder the table columns in runtime using drag-and-drop. More information is available in the section Configuring reordering of the columns (Page 392).

Access protection in runtime

To configure access protection in runtime, follow these steps:

- Activate the "Operator control - allow" property under "Properties > Properties > Miscellaneous > Toolbar > Elements > [1] Button > Operator control - allow".
- Define the type of authorization in the "Static value" column under "Properties > Properties > Miscellaneous > Toolbar > Elements > [1] Button > Authorization".

A user with the required authorization can acknowledge and edit alarms using the buttons in the alarm control.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Configuring the information bar

The information bar of the alarm control shows you the current time and the connection status, for example.

To configure the information bar, follow these steps:

1. Configure the general properties of the information bar, such as the font and background color, under "Properties > Properties > Miscellaneous > Information bar".
2. Configure the display of the information bar elements under "Properties > Properties > Miscellaneous > Information bar > Elements".

Toolbar

You can define the buttons of the alarm control in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To display additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the alarm control:

Button		Function
	Show active alarms	Shows the currently active alarms.
	Show logged alarms	Shows the logged alarms.
	Show and update logged alarms	Updates the logged alarms and shows them.
	Show defined alarms	Shows the alarms configured in the system.
	Alarm statistics - view	Visualizes statistical information, such as frequency and display duration of logged alarms.
	Alarm annunciator	Not supported in WinCC Unified.
	First line	Selects the first of the active alarms. The visible area of the alarm control is moved. This button is only enabled if the "Show recent" function is disabled.

Button		Function
	Previous line	Selects the previous alarm in relation to the currently selected alarm. The visible area of the alarm control is moved. This button is only enabled if the "Show recent" function is disabled.
	Next line	Selects the next alarm in relation to the currently selected alarm. The visible area of the alarm control is moved. This button is only enabled if the "Show recent" function is disabled.
	Last line	Selects the last of the active alarms. The visible area of the alarm control is moved. This button is only enabled if the "Autoscroll" function is disabled.
	Move to next acknowledgeable alarm	Selects the next alarm in relation to the currently selected alarm. The visible area of the alarm control is moved. This button is only enabled if the "Autoscroll" function is disabled.
	Previous page	Navigates to the next page
	Next page	Moves to the previous page
	Single acknowledgment	Acknowledges an individual alarm. A counter shows how many alarms are unacknowledged. The counter includes all connected servers, but no filters.
	Group acknowledgment	Acknowledges all active visible alarms in the alarm control that require acknowledgment, unless they are subject to single acknowledgment.
	Single confirm	Resets the alarm. Relevant for alarms with the state machine "Alarm with acknowledgment and confirmation" that have already been acknowledged and are outgoing.
	Alarms - show current	Defines whether the current alarm is always selected in the alarm control. Button not pressed: The "Show recent" function is active. • The current alarms in the alarm control are always displayed first. • The visible area of the alarm control is shifted automatically, as needed. • You cannot select the alarms individually or sort them by column. Button pressed: The "Show recent" function is paused.
	Infotext - configuration	Opens a dialog to display an infotext.
	Comment - configuration	Opens a dialog for adding a comment.
	Alarm statistics - configuration	Opens the dialog to change the time range for the alarm statistics.
	Disable alarm	Disables an alarm in the current alarm list and in the alarm log lists. The alarm is added to the display "Disabled alarms."
	Enable alarm	Shows an alarm once again.

Button		Function
	Shelve alarm	Shelves an alarm, for example, to prevent an alarm message from impairing the effectiveness of your system. The alarm appears in the "Shelved alarms" display.
	Unshelve alarm	Unshelves the respective alarm.
	Copy lines	Copies the selected alarms.
	Time base - configuration	Opens a dialog for setting the time zone for the time information shown in alarms.
	Selection display	Opens a dialog for filtering alarms. You can define the filter criteria or filter the alarms by criteria defined in the Engineering System.
	Sorting setup	Opens a dialog for setting user-defined sort criteria for the displayed alarms.
	Display options - configuration	Opens a dialog for configuring the display options of the alarm control. You can define which alarms are displayed, for example, only shelved alarms or all alarms.
	Locked alarms - configuration	Opens a dialog for configuring the display options of the locked alarms.
	Export	Starts exporting the alarms to a CSV file.
	Select context	Opens the configuration dialog of the context.

See also

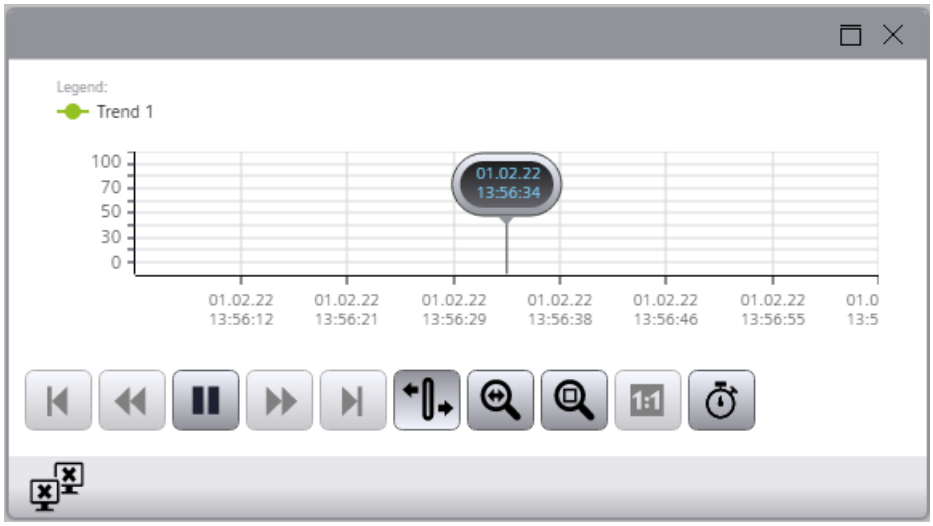
Configuring an alarm control (Page 753)

Automatically filling in of property values for an object collection (Page 379)

4.2.4.3 Trend control

Use

You can use the "Trend control" object to display tag values from the current process or from the log in the form of trends as an autorepeat.



Note

Trend display in future time range

The trend area located in the future continues the last drawn value.

Layout

In the Inspector window, you can customize the position, geometry, style, colors, and font types of the object. You can adapt the following properties in particular:

- "Trend areas": Specifies the representation of the trends.
- "Trends": Defines the configuration of the trends.
- "Toolbar": Defines the buttons for the trend control.

Configure trend area

To configure the trend display, follow these steps:

1. Create the following under "Properties > Properties > General > Trend areas":
   - Common or individual trend areas
   - Common or separate axes
   - Writing direction of all trends

   By default, the first trend area [0] is already created in the object. You can create more trend areas using the selection button in the "Static value" column.
2. Configure the value axes and the time axes.
3. Open the settings of the time axis under "Properties > General > Time axes bottom > Time axis [0]".
   Configure the "Time range" of the trend display:
   - "Time interval": You define the time range using a starting time and a following time interval.
   - "Start time and end time": You define the time range using a starting time and an end time.
   - "Measuring points": You define the time range using a starting time and a number of measuring points.
4. Open the settings of the value axis under "Properties > General > Trend areas > Trends > Left value axis > Value axis Y [0]":
   - If required, configure the value range, the output format, and the scaling of the value axis.
   - If required, configure the value range, the output format, and the scaling of the value axis.
5. Go to "Properties > General > Trend areas > Trends" and configure the trends for the trend area.

Configuring trends

To configure the trends for each trend area, follow these steps:

1. Select the data supply for the respective trend under "Properties > General > Trend areas > Trends > [0] Trend > Data source Y > Source":
   - "Logging tag": The trend control is supplied with values from a tag log.
   - "HMI tag": The trend control is supplied with values of a tag.
2. Select the data supply for the tag under "Properties > General > Trend areas > Trends > Trend [0] > Data source Y > Tag".
   - In the case of an HMI tag, specify the tag name in the "Static value" column.
   - In the case of a logging tag, enter the name of the HMI tag in the "Static value" column first.
     Enter the name of the associated logging tags separated by a colon, for example, "HMITag_1:LoggingTag_1".
3. Configure the display mode for trends under "Trend mode".

Note

In a trend control with multiple trends, a trend can also be selected using the legend in runtime.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Graphic - pressed button
- Icon

Toolbar

You can define the buttons of the trend control in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To display additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the trend control:

Button	Name	Function
	First record	Shows the trend direction starting with the first logged value.
	Previous record	Shows the trend direction of the previous time interval.
	Start/Stop	Stops and starts the trend update. Started: The trend is continuously updated. It always shows the latest values. Stopped: New values are buffered and updated as soon as you start the trend update again.
	Next record	Shows the trend direction of the next time interval.
	Last record	Shows the trend direction up to the last logged value.
	Previous trend	Displays the previous trend in the foreground.
	Next trend	Displays the next trend in the foreground.

Button	Name	Function
	Ruler	Determines the coordinates of a point of the trend.
	Zoom time axis +/-	Enlarges or reduces the time axis display.
	Zoom value axis +/-	Enlarges or reduces the value axis display.
	Zoom area	Increases the size of any section of the trend window.
	Zoom +/-	Enlarges or reduces the view in the trend window.
	Move trend area	Moves the display in the trend area. Values from the future trend area apply the last displayed value.
	Move axes area	Moves the display in the axes area.
	Original view	Switches from the magnified trend control back to the normal view.
	Select time range	Opens the dialog for setting the time range displayed in the trend window.
	Select trends	Opens the dialog for setting the visibility of trends.
	Select data connection	Opens the dialog for selecting the logs and tags to serve as the data source for the trend control.
	Statistics area	Enables you to define a time range for which statistical values are determined. Vertical lines which you use to set the time range are displayed in the trend window. To display the values, connect the trend control to the trend companion.
	Calculate statistics	Opens a statistics window to display the minimum, maximum, means, and standard deviation for the selected time range and the selected trend. To display the values, connect the trend control to the trend companion.
	Print	Starts printing the trends shown in the trend window.

Button	Name	Function
	Export	Opens the dialog for saving the trend data in CSV format.
	Select context	Opens the configuration dialog of the context.

See also

Configuring a trend control (Page 680)

Configuring the toolbar and information bar (Page 691)

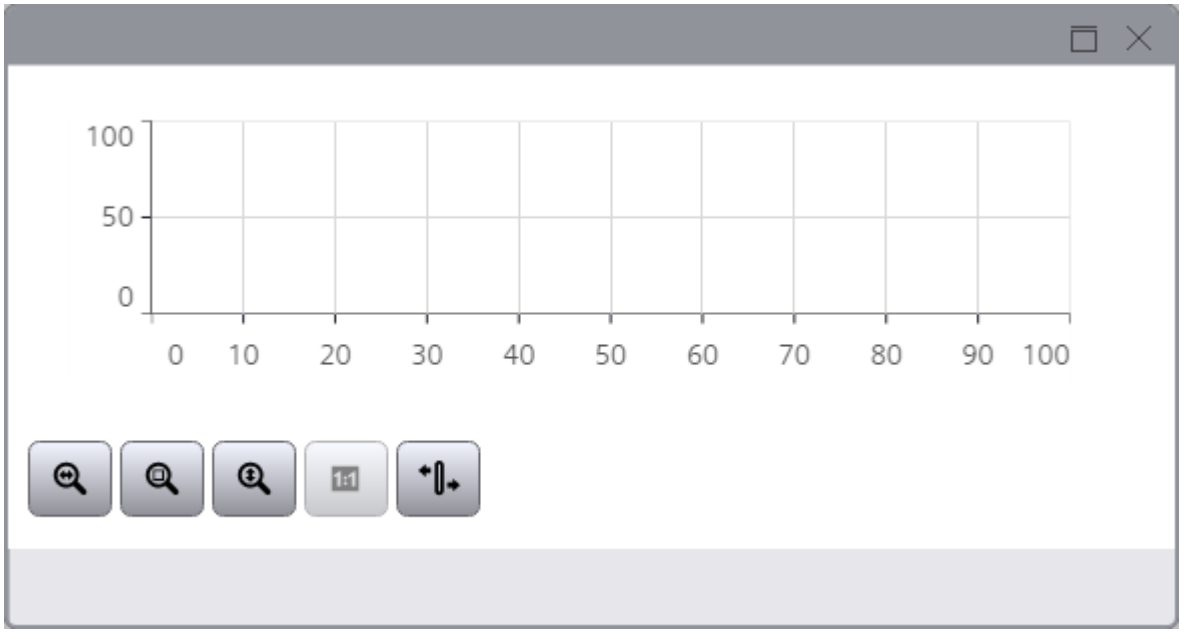
Defining the data source (Page 692)

Automatically filling in of property values for an object collection (Page 379)

4.2.4.4 Function trend control

Use

You can use the "Function trend control" object to represent the values of a tag as a function of another tag. This means that you can present temperature trends as a function of the pressure, for example. You can also compare the trend to a setpoint trend.



Note

Trend display in future time range

The trend area located in the future continues the last drawn value.

Layout

In the Inspector window, you can customize the position, geometry, style, colors, and font types of the object. You can adapt the following properties in particular:

- "Function trends": Defines the configuration of the function trends.
- "Toolbar": Defines the buttons of the function trend control.

Configuring function trends

To configure the function trends for each function trend area, follow these steps:

1. Select the data supply for the function trend under "Properties > General > Function trend - area > Function trends > [0] Function trend > Data source X > Source".
   - "Logging tag": The trend control is supplied with values from a tag log.
   - "HMI tag": The trend control is supplied with values of a tag.
2. Enter the tag name under "Properties > General > Function trend - areas > [0] Function trend - area > Function trends > [0] Function trend > Data source X > Tag":
   - In the case of an HMI tag, specify the tag name in the "Static value" column.
   - In the case of a logging tag, enter the name of the HMI tag in the "Static value" column first. Enter the name of the associated logging tags separated by a colon, for example, "HMITag_1:LoggingTag_1".
3. Configure the data supply for "Data source Y".

4. Open the settings of the time axis under "Properties > General > Function trend - area > Function trends > [0] Function trend > Time range".
   Configure the trend display for the "Time range".
   - "Time interval": You define the time range using a starting time and a following time interval.
   - "Start time and end time": You define the time range using a starting time and an end time.
   - "Measuring points": You define the time range using a starting time and a number of measuring points.
5. Configure the value range of the trend display under:
   - "Properties > General > Function trend - areas > [0] Function trend - area > Value axes - left > [0] Value axis Y".
   - "Properties > General > Function trend - areas > [0] Function trend - area > Value axes - bottom > [0] Value axis X".

   Select one of the options:
   - "Automatically adapt value range". The displayed value range is automatically adapted to the current values.
   - "Scale value - maximum" and "Scale value - minimum". Define the minimum value and maximum value for the value range.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Graphic - pressed button
- Icon
- Marker graphic

Toolbar

You can define the buttons of the function trend control in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To display additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the function trend control:

Button	Name	Function
	Start/Stop	Stops and starts the trend update. Started: The trend is continuously updated. It always shows the latest values. Stopped: New values are buffered and updated as soon as you start the trend update again.
	Zoom X axis +/-	Zooms in to or out of the X axis in the trend window.
	Zoom area	Increases the size of any section of the trend window.
	Zoom +/-	Enlarges and/or shrinks the trends in the trend window.
	Zoom Y axis +/-	Enlarges and/or reduces the Y axis in the trend window.
	Original view	Switches from the magnified trend control back to the normal view.
	Previous trend	Displays the previous trend in the foreground.
	Next trend	Displays the next trend in the foreground.
	Ruler	Determines the coordinates of a point of the trend.
	Move trend area	Moves the trends along the X axis and Y axis in the trend window. Values from the future trend area apply the last displayed value.
	Move axes area	Moves the trends along the value axis in the trend window.
	Select time range	Opens the dialog for setting the time range displayed in the trend window.
	Select trends	Opens a dialog for setting the visibility of trends.
	Select data connection	Opens a dialog for selecting logs and tags.

Button	Name	Function
	Print	Starts printing the trends shown in the trend window. The print job used during printing is defined in the configuration dialog in the "General" tab.
	Export	Starts the export of all or the selected runtime data to a "csv" file.

See also

Configuring the function trend control (Page 682)

Configuring the toolbar and information bar (Page 691)

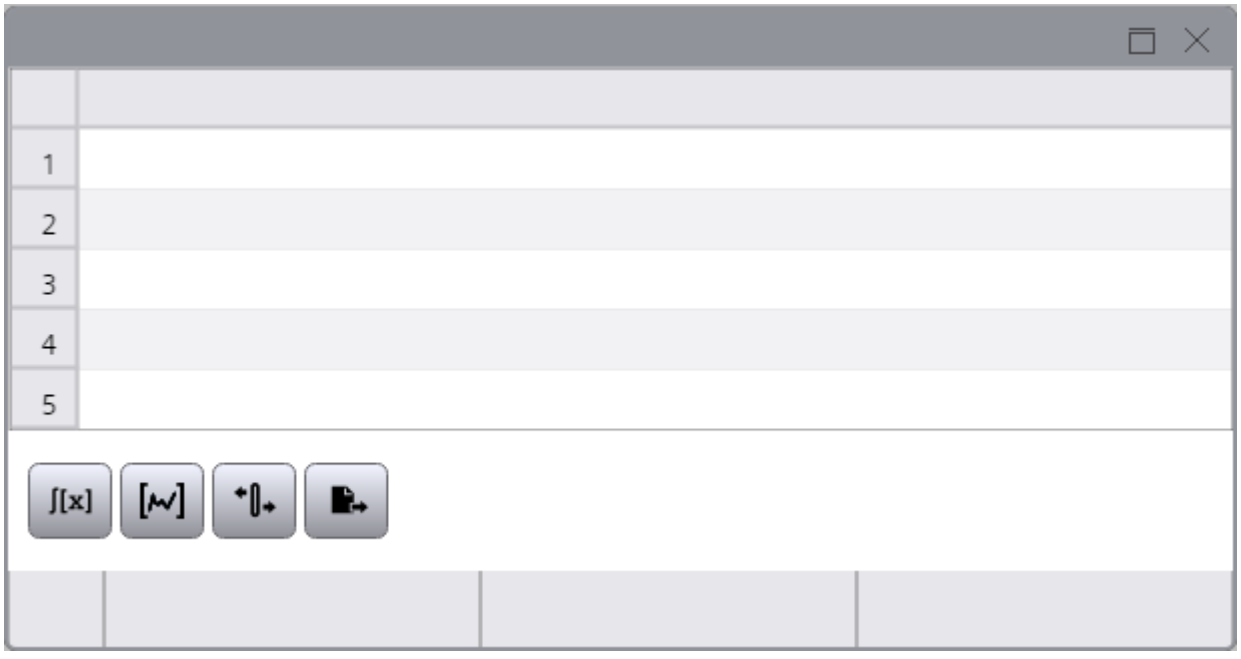
Defining the data source (Page 692)

Automatically filling in of property values for an object collection (Page 379)

4.2.4.5 Trend companion

Use

You use the "Trend companion" object to show evaluated data and statistics from a trend control or function trend control in a table.



Layout

In the Inspector window, you can customize the position, geometry, style, colors, and font types of the object. You can adapt the following properties in particular:

- "Data source": Specifies the source for representing the values.
- "Trend companion - mode": Defines the mode of the values display in the trend companion.
- "Toolbar": Specifies the buttons in the trend companion.

Defining the data source for displaying the values

To define the display of values in the trend companion, follow these steps:

1. Configure a trend control or a function trend control.
2. Select the trend companion. Click "Properties > General > Data source".
3. Select the trend control or function trend control as the data source.

To adapt the display to the connected object, select the "Data source - use background color" and "Data source - use font color" options under "Properties > Format".

By default, the format of the connected object is adopted during the configuration for the display format. The size, value range and zoom factor of the object are taken into account to display the optimum number of decimal places.

You can configure the display formats for individual values in the Inspector window of the trend companion, for example, to display an exact number of decimal places.

Defining the mode of the trend companion

To define the mode of the trend companion, follow these steps:

1. Select the mode under "Properties > Properties > General > Trend companion - mode".
2. Select one of 3 different mode types depending on the data source:
   - The "Ruler" mode shows the coordinate values of the trends on the ruler or the values of a selected row in the table.
   - The "Statistics area" mode shows the values of the low limit and high limit of the trends between two rulers or the selected area in the table.
     The "Statistics area" mode is not available for the function trend control object.
   - The "Statistic result" mode shows the statistical evaluation of the trends between two rulers or the selected values in the table.
     The "Statistics result" mode is not available for the "function trend control" object.

Configuring reordering of the columns

Configure whether operators can reorder the table columns in runtime using drag-and-drop. More information is available in the section Configuring reordering of the columns (Page 392).

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Toolbar

You can define the buttons of the trend companion in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To display additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the trend companion:

Button	Brief description	Description
	Calculate statistics	Shows the statistical values in the statistics window. The displayed values refer to a selected trend with the configured calculation time period. The button is only enabled if a statistics window is connected with a trend control.
	Statistics area	Enables you to define a time range for which statistical values are determined.
	Ruler window	Queries the coordinate points of a trend. The trend data are displayed in the ruler window.
	Print	Reserved for future versions.
	Export	Starts the export of all or the selected runtime data to a "csv" file.

See also

Configuring the trend companion (Page 689)

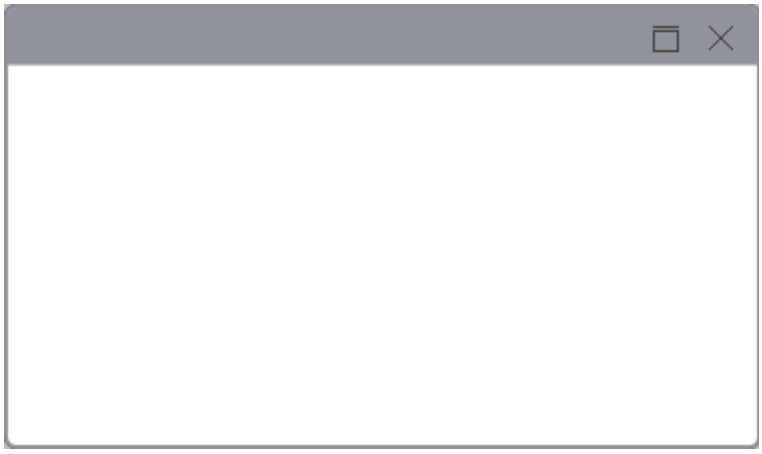
Automatically filling in of property values for an object collection (Page 379)

4.2.4.6 Screen window

Use

You can use the "Screen window" object to represent other screens from the project in the current screen. To constantly update the content of a screen window, for example, you dynamize the object.

You can also use independent screen windows independently of the screen in question. With appropriate hardware equipment and support by the operating system you can also control multiple monitors and map processes in a more comprehensive and differentiated manner.



Layout

In the Inspector window, you can customize the settings for the position, geometry, style, and color of the object. You can adapt the following properties in particular:

- "Zoom - factor": Defines the size of the embedded screen.
- "Size - fit": Specifies whether:
  - The embedded screen is scaled to the size of the screen window.
  - The screen window is scaled to the size of the embedded screen.

Matching the size of the embedded screen and screen window

To match the size of the embedded screen to the size of the screen window, choose one of the following options:

- To reduce the size of the embedded screen:
  Set the desired zoom factor in the Inspector window under "Properties > Format > Zoom - factor".
- To scroll to a section of the embedded screen:
  Enable the visibility of the horizontal and vertical scroll bars in the Inspector window under "Properties > Format". Set the position of the scroll bars.
  The user can move to details of the embedded screen in runtime.
- To adapt the embedded screen to the size of the screen window or vice versa:
  Select either "Fit window to screen" or "Fit screen to window" in the Inspector window under "Properties > Format > Size - fit".

Resizing in runtime

To allow users to resize a screen window in runtime, follow these steps:

1. Activate the options "Show border" and "Can be sized" in the Inspector window "Properties > Appearance > Window settings".
   The width of the border is not evaluated.

If the embedded screen is larger than the screen window, you can configure the scroll bars for the screen window under "Properties > Format".

Moving screen window in runtime

To allow users to move a screen window in runtime, follow these steps:

1. Activate the options "Show heading" and "Can be moved" in the Inspector window "Properties > Appearance > Window settings".

Scaling and moving screen windows in runtime with two-finger gesture

To allow users to scale the screen window on touch devices with a two-finger gesture in runtime, or to move the displayed section with a two-finger gesture, follow these steps:

1. In the "Properties > Format" Inspector window, activate the "Zoom - allow" property. Default setting: Activated

Clicking on an object behind the screen window in runtime

Screen windows can overlay other objects placed on the screen that users must operate in runtime.

To allow users to click an object placed below the screen window in runtime, follow these steps:

1. Under "Properties > General > Screen", select the screen you want to load into the screen window, for example, "Screen_2".
2. Set the "Transparent" entry in the "Static value" column under "Appearance > Background - fill pattern" in the properties of the screen selected in step 1.

Application example: You use a round menu with a button next to it. To prevent the rectangular border surrounding the menu from obscuring the button, make the menu available in a screen window. Proceeding as described above has the result that the button remains operable for users.

Dynamization of graphic properties with tags or scripts

You can dynamize the following property containing a graphic with a tag or with a script:

- Icon

4.2.4.7 Faceplate container

Use

The faceplate container is used to display faceplates in runtime. If a faceplate type has been instantiated in the container, the desired faceplate type is specified in the "Contained type" property.

You can find detailed information on configuring faceplates in the section "Configuring faceplates (Page 524)".

Layout

You can change the settings for the position, geometry, style, color, and font of the object in the Inspector window. You can adjust the following properties in particular:

- "Window settings": Specifies the representation of the faceplate container in runtime.
- "Faceplate type": Defines the faceplate type that is instantiated in the faceplate container.

Defining window settings

To resize a faceplate instance in runtime, activate the options "Show border" and "Can be sized" in the Inspector window "Properties > Appearance > Window settings".

The width of the border is not evaluated.

To move a faceplate instance in runtime, select the options "Show heading" and "Can be moved" in the Inspector window under "Properties > Appearance > Window settings".

Defining the faceplate type

To define the faceplate type, select the faceplate type that is instantiated in the faceplate container under "Properties > Miscellaneous > Faceplate type".

Dynamization of graphic properties with tags or scripts

You can dynamize the following property containing a graphic with a tag or with a script:

- Icon

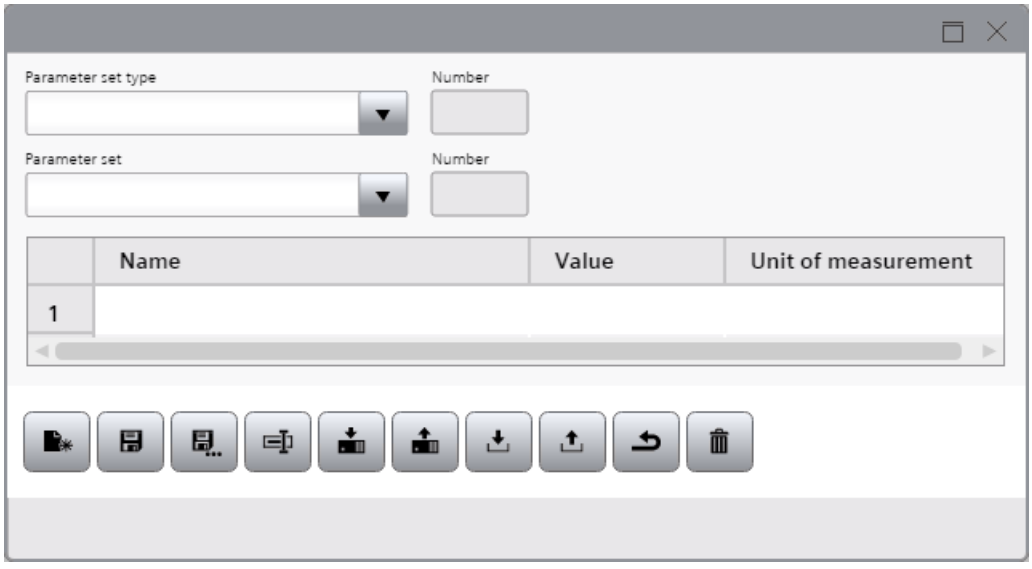
See also

Basics of faceplates (Page 524)

4.2.4.8 Parameter set control

Use

You can use the "Parameter set control" object to display and manage parameter sets in runtime and to exchange them with the controller.



Layout

You can change the settings for the position, geometry, style, color, and font of the object in the Inspector window. Under "Miscellaneous", you can adapt the following properties in particular:

- "Parameter - view": Specifies the representation of the parameter table in the object.
- "Editing mode": Defines the activation status of the toolbar buttons.
- "Information bar": Specifies the representation of the information bar.
- "Toolbar": Defines the buttons of the parameter set control.

Using a parameter set type

If you only want to use a particular parameter set type with its parameter sets in runtime, select the desired parameter set type under "Properties > General > Fixed parameter set type".

Configuring the time zone

To configure the time zone, follow these steps:

Under "Properties > Miscellaneous > Time zone", set the desired time zone by entering a numerical value.

The numerical value stands for a time zone, for example:

- "-1" stands for UTC-1h (Central European Time, standard time)
- "1" stands for UTC-12h (International Date Line West)
- "2" stands for UTC-11h (Hawaii)

Defining the editing mode

To specify the editing mode and to enable or disable the buttons, follow these steps:

Under "Properties > Miscellaneous > Editing mode", configure the activation status of the toolbar buttons "Create", "Save", "Save as", "Rename" and "Delete". These toolbar buttons are used to edit parameter sets.

You can select between the following settings:

- "None": Deactivates all buttons.
- "Update": Activates the "Save" and "Rename" buttons.
- "Create": Activates the "Create" and "Save as" buttons.
- "Delete": Activates the "Delete" button.

Configuring reordering of the columns

Configure whether operators can reorder the table columns in runtime using drag-and-drop. More information is available in the section Configuring reordering of the columns (Page 392).

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Configuring the information bar

To configure the information bar, follow these steps:

1. Configure the general properties of the information bar, such as the font and background color, under "Properties > Miscellaneous > Information bar".
2. To adjust the height of the "Status text" element, specify the height under "Properties > Miscellaneous > Information bar > Elements > [0] Element".

The "Status Text" element is the only status line element of the parameter set control. Status messages are displayed in this element in runtime.

Toolbar

You can define the buttons of the parameter set control in runtime and their operator authorizations in the Inspector window under "Properties > Miscellaneous > Toolbar > Elements". By default, all buttons are displayed in the toolbar. To hide specific buttons, deactivate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the parameter set control:

	Button	Function
	Create	Creates a new parameter set.
	Save	Saves a parameter set.
	Save as	Saves an existing parameter set under a new name and new ID.
	Rename	Renames the selected parameter set.
	Write to PLC	Writes the values of the selected parameter set to the PLC.
	Read from PLC	Writes the values of the selected parameter set from the PLC.
	Import	Imports parameter sets from a "*.tsv" file.
	Export	Exports parameter sets to a "*.tsv" file.
	Cancel	Cancels the process.
	Delete	Deletes the selected parameter set.

Note

A "*.tsv" file is a text file that uses the tabulator as a list separator.

See also

Configuring the parameter set view (Page 880)

Automatically filling in of property values for an object collection (Page 379)

4.2.4.9 System diagnostics display

Use

You can use the "System diagnostics control" object to display the diagnostic status of several PLCs using traffic light SVGs. The diagnostic status contains the overall status of all relevant PLCs. The merged state is always the worst state of all PLCs.

Layout

You can change the settings for the position, geometry, style, color, and font of the object in the Inspector window. You can adjust the following properties in particular:

- "Diagnostic view": Defines various properties for the display of system diagnostics, such as the background color and row height.
- "Information bar": Specifies the representation of the information bar.
- "Toolbar": Specifies the buttons of the system diagnostics control.

Access protection in runtime

You can configure access protection with the properties "Operator control - allow" and "Authorization" under "Properties" in the Inspector window. A logged-in user having the required authorization can acknowledge and edit the system diagnostics control using the buttons in the system diagnostics control.

Defining the properties of the system diagnostics control

To define the properties of the system diagnostics view, follow these steps:

1. Click "Properties > General > Diagnostic view" in the Inspector window.
2. Define the settings for the rows and cells.
   - "Row height": Defines the height of the rows in the alarm control.
   - "Cells - internal spacing": Defines the internal spacing in the cells.
3. Define the settings for the headers under "Properties > General > Diagnostic view > Header - settings":
   - "Row header": Defines whether each row has a header.
   - "Column header": Specifies the representation of the column header.
4. Define the width and color of the grid lines.
5. Define the use of scroll bars.

Setting up column sorting

To set up the column sorting, follow these steps:

1. In the Inspector window, click "Properties > General > Diagnostic view > Columns > [0] Column".
2. Select the sorting direction and sorting order for the individual columns.

Configuring reordering of the columns

Configure whether operators can reorder the table columns in runtime using drag-and-drop. More information is available in the section Configuring reordering of the columns (Page 392).

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Configuring the information bar

The information bar of the system diagnostics control shows the connection status and path.

To configure the information bar, follow these steps:

1. Configure the general properties of the information bar, such as the font and background color, under "Properties > Miscellaneous > Information bar".
2. Configure the display of the information bar elements under "Properties > Miscellaneous > Information bar > Elements".

Toolbar

You can define the buttons of the system diagnostics control in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To display additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the system diagnostics control:

Button		Function
	Home	Shows the home page.
	Reload	Updates the view of the diagnostic event.
	First line	Selects the first of the pending diagnostic events. The visible area of the view is moved.
	Previous line	Selects the previous diagnostic event, starting from the currently selected diagnostic event. The visible area of the view is moved.
	Next line	Selects the next diagnostic event, starting from the currently selected diagnostic event. The visible area of the view is moved.
	Last line	Selects the last of the pending diagnostic events. The visible area of the view is moved.
	Share view	Enables/disables the detail view.
	Previous	Navigates to the previous PLC.
	Show diagnostic buffer	Changes from the matrix view to the diagnostic view. The diagnostic view shows the diagnostics buffer of the PLC. This button is only enabled if a PLC or one of its lower-level modules is shown in the matrix view.

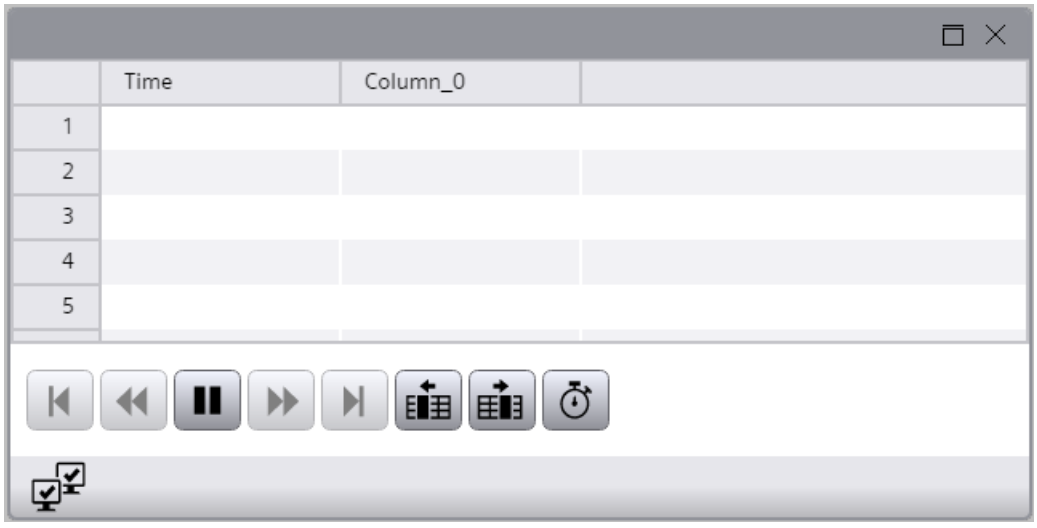
See also

Automatically filling in of property values for an object collection (Page 379)

4.2.4.10 Process control

Use

You use the "Process control" object to display the tag values in a table. You can display current, or logged values in the table. You can configure up to nine value columns. The first column is reserved for the time column.



Layout

In the Inspector window, you can customize the position, geometry, style, colors, and font types of the object. You can adapt the following properties in particular:

- "Column": Defines the setting of the value column.
- "Data source": Specifies the source for representing the values.
- "Toolbar": Specifies the buttons for the process control.

Configuring columns

To configure the columns, follow these steps:

1. Open the settings of the time column under "Properties > Miscellaneous > Process control > Columns > Time range column [0]".
2. Under "Properties > Miscellaneous > Process control > Columns > [0] Time range column > Time range", select the time range of the table:
   - "Time interval": You define the time range using a starting time and a following time interval.
   - "Start time and end time": You define the time range using a starting time and an end time.
   - "Measuring points": You define the time range using a starting time and a number of measuring points.

3. Open the settings of the respective value column under "Properties > Miscellaneous > Process control > Columns > [1] Column".
4. Under "Sort order", define the order in which the columns of the process control are shown.
5. Set the direction in which the values are sorted under "Sorting direction - default".
6. Define whether operators can re-arrange the table columns in runtime using drag-and-drop. More information is available in the section Configuring reordering of the columns (Page 392).

Defining the data source for displaying the values

To define the display of the values in the process control, follow these steps:

Under "Properties > Miscellaneous > Process control > Columns > [1] Column > Data source > Source", select:

1. Type of data source:
   - HMI tag
   - Logging tag
2. Tag that supplies the column with values.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Toolbar

You can define the buttons of the process control in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To add more buttons in the display object, activate the "Visibility" property under the settings of the corresponding button.

The following buttons are available for the process control:

Button	Name	Function
	First record	Shows the tag values starting with the first logged value.
	Previous record	Shows the tag values in the previous time interval.
	Start/Stop	Stops and starts the column update. The values are buffered and updated as soon as you start column update again.

Button	Name	Function
	Next record	Shows the values of the tag in the next time interval.
	Last record	Shows the tag values up to the last logged value.
	Edit	Allows the editing of data in any table field that is opened when the user double-clicks it.
	Previous column	Displays the previous column in the foreground
	Next column	Displays the next column in the foreground
	Select time range	Opens the dialog for setting the time range displayed in the process control.
	Select data connection	Opens the dialog for selecting the archives and tags that serve as data the source for this process control.
	Create archive value	Creates an archived value.
	Delete log value	Deletes a logged value.
	Export	Starts the export of all or the selected runtime data to a "csv" file.

See also

Configuring the process control (Page 688)

Automatically filling in of property values for an object collection (Page 379)

4.2.4.11 Web control

Use

You use the "Web control" object to display basic HTML pages and documents in PDF format.

You have access to the data of the local user management in runtime via a "Browser".

Layout

Customize the object position and size in the Inspector window. In particular, you can customize the following property:

- "URL": Specifies which Internet address is opened in the HTML Browser.
- "Toolbar": Specifies the buttons of the browser.

Defining the URL

The "browser" object supports the following protocols:

- On a Unified Comfort Panel, HTTP protocol and HTTPS protocol.
- On a Unified PC, only the HTTPS protocol.

To define the URL, follow these steps:

Define the Internet address in the Inspector window under "Properties > Properties > URL".

Displaying HTML pages

Please note the following when using the object:

- The "Web control" object only displays content that is supported by the web browser in which runtime is open.
- The object is implemented as an iFrame. Pages with X-frame option settings that prevent the display in an iFrame are not displayed in the object.

Limitations

The "Web control" object has a limited range of functions compared to a standard browser:

- Navigation from the "Web control" object is not supported (top-level navigation).
- Calls of queries and dialogs (popups and modal dialogs) are only supported if they were activated in the file <Path for the WinCC Unified installation directory>WinCCUnified\WebRH\public\content\custom\CustomSettings.json:
  `{"CustomSettings": {"HmiWebControl" : {"AllowPopups" : true,"AllowModals" : true}}}`

  Note

  Popups and modal dialogs stop the update.

Displaying PDF files in the "Browser" on a Unified PC

The "Browser" object displays PDF files that are available:

- Locally on the HMI device
- On the Internet

You can view a PDF file in the following ways:

- Copy the PDF files to the directory "C:\Program Files\Siemens\Automation\WinCCUnified\WebRH\public".
  Under "Properties > URL", enter the address "https://localhost/WebRH/<pdfname.pdf>".

  Note

  You cannot display any PDF files that are saved locally in a different directory on your PC.

  You can also use the IP address or the PC name instead of "localhost".
  If you operate runtime on a different PC than the TIA Portal, also save the PDF files on the runtime PC.
- Enter a valid Internet address under "Properties > Properties > URL".

Influencing how the document is displayed on a Unified PC

The "Browser" object supports a large number of default parameters with which you can influence how a PDF file is displayed.

Examples of parameters when opening the PDF file:

- Jump to specific page: https://winccunified/WebRH/UCPManual.pdf#page=18
- Jump to table of contents: https://winccunified/WebRH/UCPManual.pdf#Inhaltsverzeichnis
- Zoom in on page: https://winccunified/WebRH/UCPManual.pdf#zoom=200

Displaying PDF files in the "Browser" on a Unified Comfort Panel

The "Browser" object displays PDF files that are available:

- Locally on the HMI device
- On an external storage medium

You can view a PDF file in the following ways:

- Enter path and file name in the URL input field of the "Browser" operating object.
- In the configuration of the "Browser" operating object under "Properties", link the URL with a tag of the type WString which contains path and file name.

Syntax: file:///<path>/<filename>.pdf

Pay attention to uppercase/lowercase spelling.

Examples:

- Open file from the data memory card: file:///media/simatic/X51/UCPManual.pdf
- Open locally saved file: file:///home/industrial/UCPManual.pdf

Influencing how the document is displayed on a Unified Comfort Panel

The "Browser" object supports a large number of default parameters with which you can influence how a PDF file is displayed.

Examples of parameters when opening the PDF file:

- Open file on page 20: file:///media/simatic/X51/UCPManual.pdf?20#page=20
- Open file with zoom factor 150%: file:///media/simatic/X51/UCPManual.pdf?150#zoom=150
- Open file on page 20 with zoom factor 150%: file:///media/simatic/X51/UCPManual.pdf?(20,150)#page=20&zoom=150

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Toolbar

You can define the buttons of the browser in runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To add more buttons in the display object, activate the "Visibility" property under the settings of the corresponding button.

See also

Automatically filling in of property values for an object collection (Page 379)

4.2.4.12 Media Player

Use

You use the "Media Player" object to play multimedia files in Runtime.

Layout

You can set the following properties in the Inspector window:

- "URL": Defines the file that is played back in the Media Player.
- "Toolbar": Specifies the buttons of the Media Player.

Supported file formats

The Media Player supports all file formats that are supported by the utilized browser.
The playback of multimedia files in the object depends not only on the file format, but also on the video and audio codecs installed on the computer.

Note

If you copy the project to another PC, keep the following in mind: Files specified in the WinCC Media Player are not copied with the project if the files are dynamically linked and no UNC path is specified. You have to load the files into the project again.

Accessing a file in the Media Player

To access files in the Media Player, you must have stored them in the Public directory of the WinCC Unified installation.

To access a file in the Media Player, follow these steps:

1. Store the file in the Public directory of the WinCC Unified installation, e.g. "C:\Program Files\Siemens\Automation\WinCCUnified\WebRH\public". You can also create a subdirectory, e.g. "MediaFiles".
2. Click "Properties > All properties > URL". Enter the URL.
   You can structure a valid URL according to the following scheme: "https://<ComputerName>.<DomainName>/WebRH/<FileName>".
   Example: "https://mycomputer.siemens.net/WebRH/Twistlock.mp4".
   If you have created a subdirectory, e.g. "MediaFiles", enter the URL in the following format: "https://mycomputer.siemens.net/WebRH/MediaFiles/Twistlock.mp4".

Alternatively, you can change the address via a script, e.g.:

```
Screen.FindItem('MediaControl').Url = https://
<ComputerName>.<DomainName>/WebRH/<FileName>
```

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Graphic - pressed button
- Icon

Toolbar

You can define the buttons of the Media Player in Runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To add more buttons in the display object, activate the "Visibility" property under the settings of the corresponding button.

The following buttons are available for the Media Player:

Button	Name	Function
	Play	Plays the video or audio file.
	Pause	Pauses the video or audio file.
	Stop	Stops the video or audio file.
	Search forward	Searches for the next video or audio file.

Button	Name	Function
	Search backward	Searches for the last video or audio file.
	Mute	Mutes the video or audio file.

See also

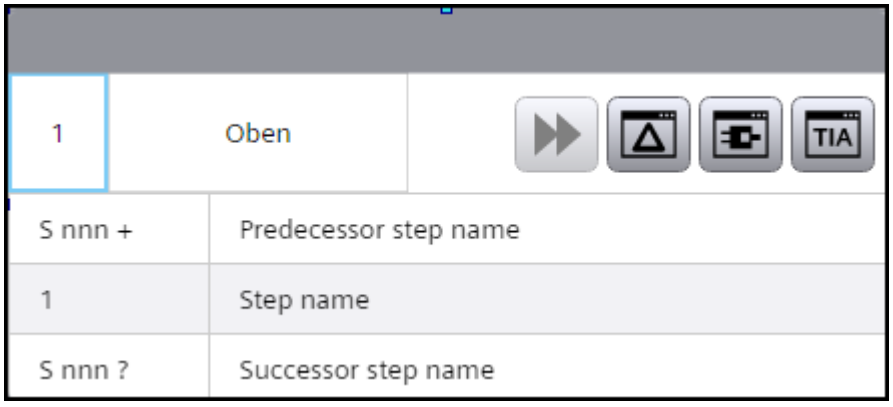
Automatically filling in of property values for an object collection (Page 379)

http://support.automation.siemens.com (http://support.automation.siemens.com/WW/view/en/62101921)

4.2.4.13 GRAPH overview

Use

The "GRAPH Overview" object is used to display the current program status for executed steps of the GRAPH sequencer. Errors during execution of a program are displayed directly at the corresponding step.



The following information is displayed in the "GRAPH Overview" object:

- Name and status of the function block
- Status of initial and simultaneous steps
- Number and name of the first step currently executed step
- Operating mode for running the GRAPH sequencer

WinCC supports the display of step names for the GRAPH blocks in multiple languages starting from Version 6.0. The step names will then be displayed in the selected Runtime language following a language changeover in Runtime. If the selected language is not available in the GRAPH block, the names are displayed in the default language (English).

Note

Device dependency of the "GRAPH Overview" object

The "GRAPH overview" object is available for Unified PC.

Note

Requirement for display in GRAPH overview

For the display of the program status of an S7 GRAPH instance data block in the "GRAPH overview" object to be possible, the instance-specific properties of the block must be set as "Visible in HMI" and "Accessible from HMI".

Layout

In the Inspector window, you customize the position, style, colors and font types of the object. You can adapt the following properties in particular:

- Assigned GRAPH DB tag
- Buttons of the toolbar

Operating mode

Four operating modes are available for running the GRAPH sequence:

- AUTO (default setting) - Automatically switches to the next step when the transition is fulfilled.
- TAP - Automatically switches to the next step when the transition is fulfilled and there is an edge change from "0" to "1" at the T_PUSH parameter.
- TOP - Automatically switches to the next step when the transition is fulfilled or there is an edge change from "0" to "1" at the T_PUSH parameter.
- MAN - The next step is not automatically enabled when the transition is fulfilled. The steps can be selected and deselected manually.

Note

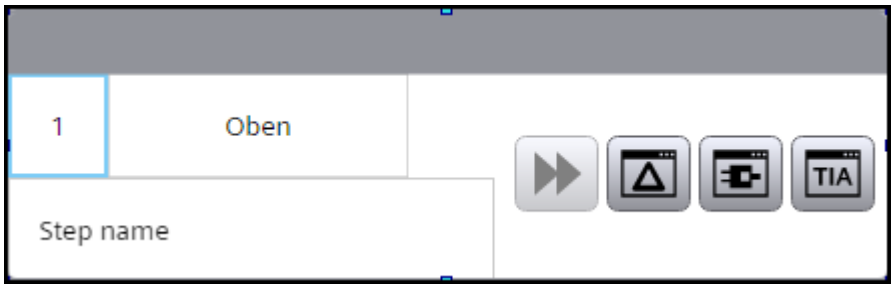
You set the operating mode by modifying the interface parameters of the GRAPH block in your control program.

In WinCC Unified Runtime, you have the option to customize the name for the operating mode that is displayed in the GRAPH overview.

Configuring a compact view

You can also configure a slim GRAPH overview without toolbar buttons and operating mode display.

To display a slim GRAPH overview in single-line compatibility mode, drag the control to the desired size.



Symbols

The symbols displayed in the GRAPH overview are pre-defined:

Symbol	Name	Function
	Error	Indicates that an error has occurred during the execution of a step.
	Initial step	Indicates that the currently executing step is the first step in the GRAPH block.
	Simultaneous step	Shows that there are other simultaneous steps in the GRAPH block in addition to the current one.

Buttons

You specify the buttons that are displayed in the GRAPH overview under "Properties > Miscellaneous > Toolbar > Elements".

Button	Name	Function
	Next Step	Jumps to the next step in parallel step. When you get to the last step, you can jump back to the first step.
	Jump to Alarm Control	Opens the configured alarm view with the error alarm in WinCC Unified. The button is intended to be populated with appropriate system functions/ scripts.
	Jump To PLC Code Viewer	Opens the configured PLC code view. The button is intended to be populated with appropriate system functions/ scripts. Ideally, use the "OpenViewerGraphFromOverview" system function.
	Jump to TIA Portal	Several system functions are available for opening the TIA Portal.

See also

Configuring a GRAPH overview (Page 6880)

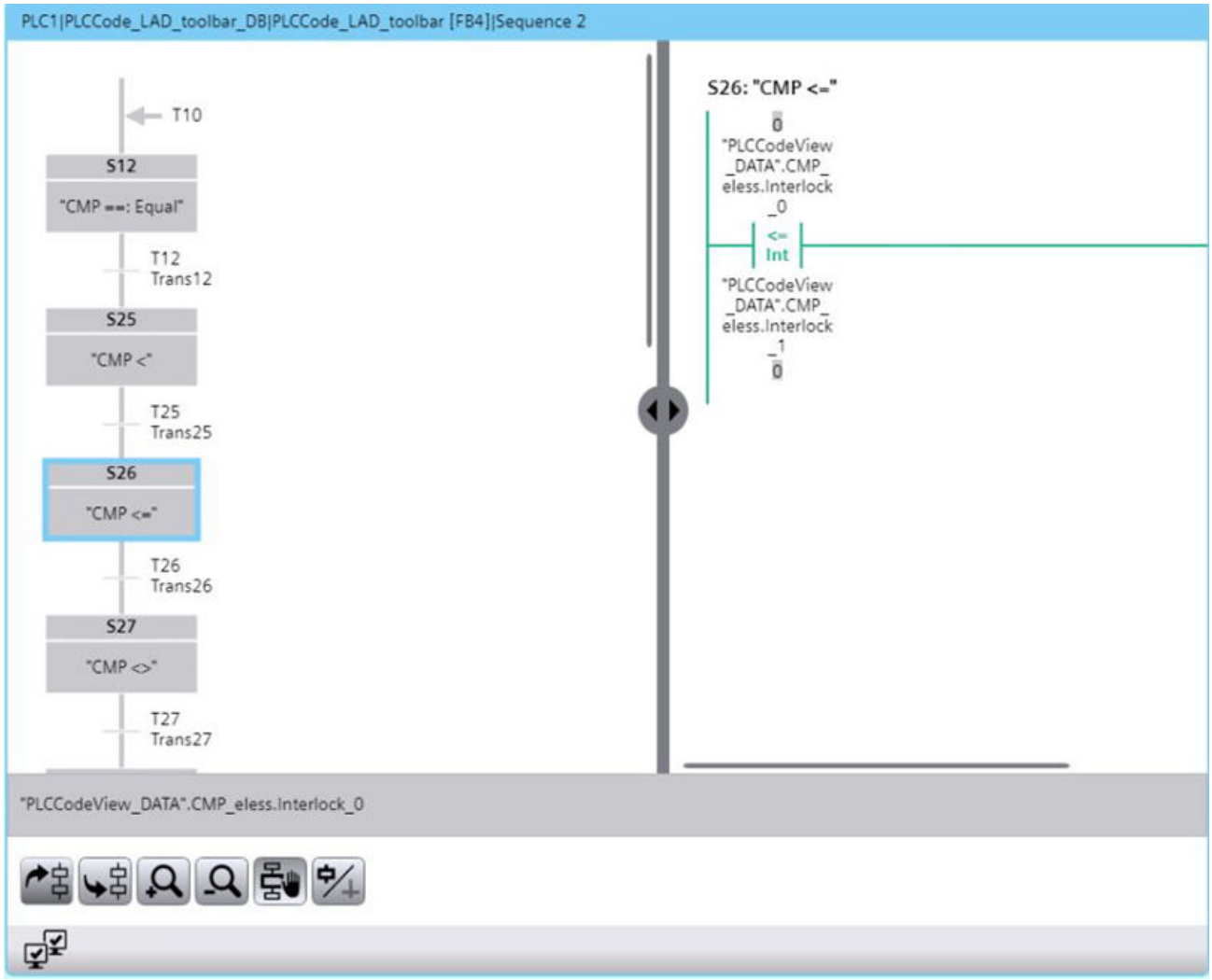
4.2.4.14 PLC code view

Use

The "PLC code viewer" object is used to display the current program status of user programs that have been programmed in the GRAPH programming language.

In the PLC code view, you display various items of information about the user program:

- Information area
- Toolbar
- Detail view
- Transition/Interlock view



Information area

The information area shows the GRAPH sequence in the left area and the details, e.g. for the step or for the transition, in the right area.

Toolbar

The toolbar shows information about the first or the selected icon.

Buttons of the toolbar

The table below shows the buttons on the toolbar and their meaning.

Operating element	Description	Function
	"Previous network"	Navigates to the previous network.
	"Next network"	Navigates to the next network.
	"Zoom in"	Enlarges the information area.
	"Zoom out"	Reduces the information area.
	"Step mode"	Switches between manual and automatic step selection for the active step.
	"Transition or Interlock"	Switches between the transition and interlock networks.

See also

Configuring the PLC code view (Page 6884)

4.2.5 My Controls

4.2.5.1 Using custom web controls

Introduction

You can use custom web controls in WinCC.

For custom web controls to be displayed in the TIA Portal, store the custom web controls in the folder <project_folder>/UserFiles/CustomControls.

Requirement

- A project has been created.
- A Unified device has been created.
- A screen is open.

Using custom web controls

Custom web controls are stored in the TIA Portal in "Tools > My Controls".

If you want to use a custom web control, follow these steps:

1. Insert the custom web control into the open screen by dragging-and-dropping the control from the "My Controls" palette to the screen.
2. Specify the properties of the custom web control in the Inspector window.
3. Configure the events for the custom web control.

Using custom web controls as a master copy

When you add the custom web control as a master copy to the project library, the control is changed.

You update the copied custom web control by clicking on the "Update" icon.

Managing properties and events

You can change the properties and create or delete events without having to open the project again or restart TIA Portal.

You have the following two options for managing the properties and events:

- Extract the manifest file from the .zip file, make the changes, and save the manifest file back to the .zip file.
- Create a separate .zip file with the changed manifest file and overwrite the .zip file in the CustomControls folder.

4.2.5.2 Updating Custom Web Controls

Introduction

After changing the properties for custom web controls or configuring new events, you can update custom web controls.

Updating custom web controls

If you have changed the properties for custom web controls or configured new events, you can update the custom web controls as follows:

1. Click the Update icon in the "My Controls" palette.
2. Custom web controls are updated.
3. A message appears in the Inspector window:
   - "The object '{0}' was updated successfully".
   - "The object '{0}' was updated successfully, but some properties have been lost due to incompatible changes to the interface."
   - "All objects are up to date". You have not changed any properties or alarms.

Note

The placeholder '{0}' stands for a unique and complete path on which the custom web control is stored.

Restrictions for the update

If you have opened a project as read-only, the update is not possible.

The following changes to the custom web control prevent the automatic update:

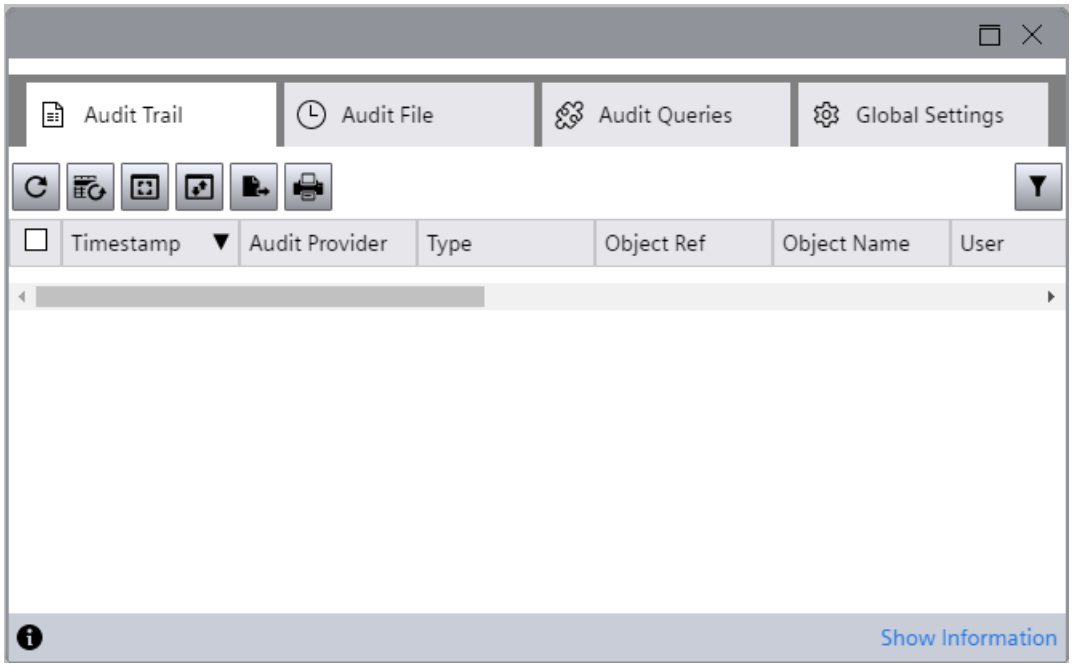
- Renaming properties or events
- Deleting properties or events
- Changing the data type

4.2.5.3 My Controls - Overview

Audit Viewer

Use

You use the "Audit Viewer" object to evaluate in table form all data of the audit trail in Runtime.



Layout

In the Inspector window, you can customize the settings for the position, geometry, style, and color of the object. You can adjust the following properties in particular:

- "Text": Specifies the text for the label.
- "Window settings": Defines the settings for display in Runtime.

Text

To set a text for the Audit Viewer, follow these steps:

1. Click "Properties > Miscellaneous > Label > Font" in the Inspector window.
2. Select a font.
3. Click "Properties > Miscellaneous > Label > Text" in the Inspector window.
4. Enter a text.

Resizing in Runtime

To resize the Audit Viewer in Runtime, follow these steps:

1. Activate the options "Show border" and "Can be sized" in the Inspector window "Properties > Appearance > Window settings".
   The width of the border is not evaluated.

Dynamization of graphic properties with tags or scripts

You can dynamize the following property containing a graphic with a tag or with a script:

- Icon

Plant overview

Use

You use the "Plant overview" object to display the configured plant view in runtime.

You use it to navigate to the plant objects within the plant structure and get an overview of your plant at one glance.

If you have configured screens or alarms for the lower-level plant objects and have linked them to the "Plant overview" object, navigate to these screens and alarms and display them.

Layout

You change the settings for the position, geometry, style, and color of the object in the Inspector window.

To enable navigation between the screens of the plant objects, configure the companion controls under "Properties > Miscellaneous > Interface > Companion controls".

Buttons

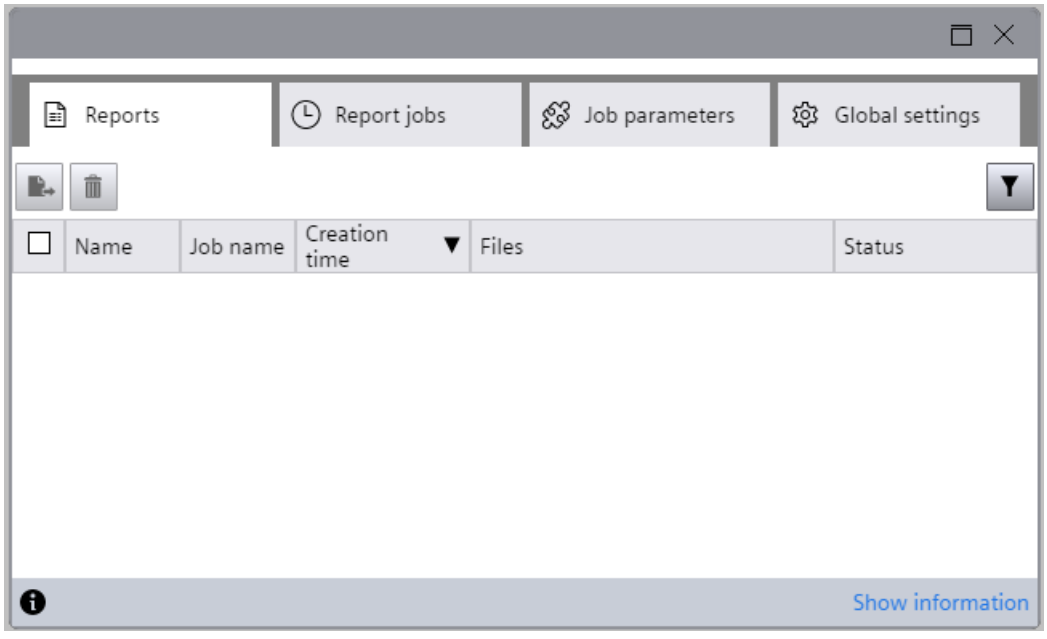
The following buttons are available for the "Plant overview" object in runtime:

Button	Name	Function
	Expand	Expands the plant view with the lower-level plant objects.
	Collapse	Collapses the plant view with the lower-level plant objects.
	Filter	Defines which plant objects are displayed.

Reports

Use

You use the "Reports" object to create and manage report tasks in runtime. You have access to the reports generated by the report jobs.



You can find detailed information on configuring the object in the engineering system in the section Configuring production reports in the engineering system (Page 7655).

You can find detailed information on configuring report tasks in runtime in the section Working with production reports in Runtime (Page 7719).

Layout

In the Inspector window, you change general settings of the object such as the position, height, width, label and window settings.

See also

Basics of Reporting (Page 7647)

The user interface of the "Reports" control (Page 7722)

4.2.6 Graphics

4.2.6.1 External graphics

Introduction

You can use graphics created with an external graphic program in WinCC. To use these graphics you store them in the project graphics of the WinCC project.

You can save graphics in the project graphics:

- When you drag-and-drop graphics objects from the "Graphics" pane into the work area, these are stored automatically in the project graphics. The graphic names are numbered in the order of their creation, for example, "Graphic_1." Use the <F2> function key to rename the graphic.
- As a graphic file with the following formats:
  *.bmp, *.ico, *.emf, *.wmf, *.gif, *.tif, *.png, *.svg, *.jpeg or *.jpg
- As an OLE object that is embedded in WinCC and is linked to an external graphic editor. In the case of an OLE link, you open the external graphic editor from WinCC. The linked object is edited using the graphic editor. An OLE link only works if the external graphic editor is installed on your PC, and supports OLE.

Note

High-resolution graphic objects require a lot of memory in the project and cause long loading times. They also reduce performance in Runtime.

Use graphic objects with a resolution that is sufficient for a high-quality display in the Runtime project. Note the display resolution of the target device and the size in which the graphic object is displayed on the display of the target device. Adapt the resolution of large graphic objects accordingly before using them in your project.

Use of graphics from the project graphics

Graphics from the project graphics are used in your screens:

- In a graphic view
- In a graphic list
- As labeling for a button

To use a graphic in the screen or in the screen object, drag-and-drop the desired graphic to the screen or the screen object. Alternatively, select the graphic from the selection list in the "Graphic" property in the Inspector window.

Transparent graphics

In WinCC, you also use graphics with a transparent background. When a graphic with a transparent background is inserted into a graphic object of WinCC, the transparency is replaced by the background color specified for the graphic object. The selected background color is linked firmly to the graphic. If you use the graphic in another graphic object of WinCC, this object is displayed with the same background color as the graphic object that was configured first. If you want to use the graphic with different background colors, include this graphic in the project graphics again under a different name. The additional background color is configured when the graphic is used at the corresponding graphic object of WinCC.

Managing graphics

An extensive collection of graphics, icons and symbols is installed with WinCC. In the Toolbox window of the "Graphic" pane the graphic objects are structured by topic in the "WinCC graphics folder." The link to the WinCC graphics folder cannot be removed, edited or renamed.

The "Graphics" pane is also used to manage the external graphics. The following possibilities are available:

- Creating links to graphics folders
  The external graphic objects in this folder, and in the subfolders, are displayed in the toolbox and are thus integrated in the project.
- Editing folder links
- You open the program required for editing of the external graphic in WinCC.

4.2.6.2 Managing external graphics

Introduction

External graphics that you want to use in WinCC are managed in the "Screens" editor by using the "Tools" task card in the "Graphics" pane.

Requirement

- The "Screens" editor is open.
- The "Toolbox" task card is open.

- The graphics are available.
- The graphics have the following formats:
  *.bmp, *.ico, *.emf, *.wmf, *.gif, *.tif, *.svg, *.jpeg, *.jpg

Creating a folder link

1. Click "My graphics folder."
2. Select "Link" in the shortcut menu.
   The "Create link to folder" dialog is opened. The dialog suggests a name for the folder link.
3. Edit the name as required. Select the path containing the graphic objects.
4. Click "OK" to confirm your input.
   The new folder link is added to the "Graphics" object group. The external graphics that are located in the target folder and in sub-folders are displayed in the toolbox.

Editing folder links

1. Select the folder link to edit.
2. Select the "Edit link..." command from the shortcut menu.
   The "Create link to folder" dialog is opened.
3. Edit the name and path of the folder link as required.
4. Click "OK" to confirm your input.

Renaming the folder link

1. Select the folder link to rename.
2. Select "Rename" from the shortcut menu.
3. Assign a name to the new folder link.

Removing a folder link

1. Select the folder link you want to delete.
2. Select "Remove" in the shortcut menu.

Edit external graphics

1. Select the graphic you want to edit.
2. Select the "Edit graphic" command from the shortcut menu.
   This opens the screen editor associated with the graphic object file.

Editing graphics folders from WinCC

1. Select the graphic you want to edit.
2. Select "Open folder" from the shortcut menu.
   The Windows Explorer opens.

4.2.6.3 Managing SVG graphics

Introduction

You can import SVG graphics into the TIA Portal for visualization. You can adapt the source file for the SVG graphics so that the SVG graphics are displayed correctly in the TIA Portal.

Requirement

- The HMI screen is open.

Importing SVG graphics

To import an SVG graphic into the TIA Portal, follow these steps:

1. In the project tree, under "Languages & Resources", click on "Project graphics".
2. Use drag-and-drop to add an SVG graphic from your storage location into the project graphics.
3. In the "Default graphic" column, click the added graphic.
4. Information about the SVG graphic is displayed in the Inspector window under "General > Properties".

Displaying SVG conversion information

In the Inspector window, under "Properties > General" in the "Graphics" area, you can find the "Show SVG conversion information" button.

To display the SVG conversion information, follow these steps:

1. Click on the button.
2. An "SVG conversion information" dialog with 2 scrollable read-only text boxes opens:
   - "SVG content": The content of the SVG image file is imported into the TIA Portal as a result of the conversion.
   - "SVG conversion protocol": Messages about the changes during conversion.
     During the import of SVG files into the TIA Portal, the contents that are not supported by WinCC Unified and HMI systems are removed.

You can select and copy the text in the dialog.

"SVG conversion information" dialog

The "SVG conversion information" dialog contains:

- Messages created during the import of the SVG file.
- Information on the content that was removed during conversion.

Depending on the exact content of the original SVG file, checking and conversion may be carried out in multiple test cycles. The protocol segments of the test cycles are separated by a hyphenated separator line ("-----"). The row number of the original SVG file is only specified for the first conversion test cycle.

The conversion protocol is created and provided in English, regardless of the current user interface language or runtime language settings in the project.

Editing SVG graphics

To edit an SVG graphic, follow these steps:

1. Right-click the graphic in the "Default graphic" column.
   The shortcut menu opens.
2. Click "Edit".
   An external editor for editing the graphics opens.

You can now edit the SVG graphic.

4.2.6.4 Restrictions on SVG graphics

Introduction

The SVG graphics support the SVG 1.2 Tiny standard. Note the following restrictions when using SVG graphics:

Restrictions on SVG graphics

- The CSS definitions are converted to inline attributes.
- Embedded scripts and non-local URLs are not supported in the SVG graphics and are removed from the original graphics during conversion when imported into the TIA Portal.
- The use of SVG graphics with embedded animations is not supported.
- The use of large SVG graphics affects performance due to the load associated with the increased characters.
- Migration of SVG graphics from WinCC V7 to TIA Portal is not supported.

- The following SVG characteristics are not supported:
  - Scripting
  - Interactivity
  - Styling
  - Expandability - no ForeignObjects
  - Animations
- Screen objects using an SVG graphic that was scaled in the Engineering System as a background graphic are not displayed correctly in Runtime in Chrome.
- SVG graphics for which the "Scale background graphic" property in engineering in faceplates is set to "Stretch to fit" are not displayed correctly in Runtime in Chrome.
  Convert such a graphic into a bitmap to use it in engineering. Or, for Runtime, select a web browser other than the web client, such as Firefox.
- If the "Scale background graphic" property is configured with the value "None" for screen objects, set the following SVG attributes to display SVG graphics:
  - Width: Pixel or percent
  - Height: Pixel or percent

  Specifications for width and height are recommended in pixels. If the width and height are not set, the pixel values entered in the SVG attribute "viewbox" are used. This does not apply to Firefox.

4.2.7 Dynamic widgets

4.2.7.1 Managing dynamic SVG graphics

Introduction

You can use dynamic SVG graphics in WinCC. For SVG graphics to be displayed in the TIA Portal, store the graphics in the folder <project_folder>/UserFiles/SVGControls.

Requirement

- A project has been created.
- A Unified device has been created.
- A screen is open.

Managing dynamic SVG graphics

The dynamic SVG graphics are stored in the TIA Portal in "Tools > Dynamic widgets > Project graphics".

If you want to manage an SVG graphic, follow these steps:

1. Insert the SVG graphic into the open screen by dragging the graphic from the "Project graphics" folder to the screen.
2. Specify the properties of the SVG graphic in the Inspector window.
3. Configure the events for the SVG graphic.

You can change the properties and create or delete events without having to open the project again or restart TIA Portal.

The SVG graphic is also displayed in the project graphics in "Project tree > Languages and resources".

Updating dynamic SVG graphics

If you have changed the properties of the SVG graphic or configured new events, you can update the SVG graphic as follows:

1. Click the Update icon  in the "Dynamic widgets" palette.
2. The dynamic SVG graphics are updated.
3. A message appears in the Inspector window:
   - "The object '{0}' was updated successfully".
   - "The object '{0}' was updated successfully, but some properties have been lost due to incompatible changes to the interface."
   - "All objects are up to date". You have not changed any properties or alarms.

Note

The placeholder '{0}' stands for a unique and complete path on which the SVG graphic is stored.

Restrictions for the update

If you have opened a project as read-only, the update is not possible.

The following changes to the SVG graphic prevent the automatic update:

- Renaming properties or events.
- Deleting properties or events.
- Changing the data type.

Using dynamic SVG graphic as a master copy

When you add the SVG graphic as a master copy to the project library, the SVG graphic is changed.

You update the copied SVG graphic by clicking on the "Update" icon.

Restrictions on SVG graphics

The SVG graphics support the SVG 1.2 Tiny standard. Note the following restrictions when using SVG graphics:

- The CSS definitions are converted to inline attributes.
- Embedded scripts and non-local URLs are not supported in the SVG graphics and are removed from the original graphics during conversion when imported into the TIA Portal.
- The use of SVG graphics with embedded animations is not supported.
- The use of large SVG graphics affects performance due to the load associated with the increased characters.
- Migration of SVG graphics from WinCC V7 to TIA Portal is not supported.
- The following SVG characteristics are not supported:
  - Scripting
  - Interactivity
  - Styling
  - Expandability - no ForeignObjects
  - Animations

If screen items have the value "None" for the "Scale background graphic" property, set the following SVG attributes to display SVG graphics:

- Width: Pixel or percent
- Height: Pixel or percent

Note

Specifications for width and height

Specification in pixels is recommended. If the width and height are not set, the pixel values entered in the SVG attribute "viewbox" are used. This does not apply to Firefox.

Note

Scaled SVG graphics in Chrome

Elements using an SVG graphic that was scaled in the engineering system as background graphic are not displayed correctly in Chrome in Runtime.

Editing dynamic SVG graphics

It is not possible to open SVG graphics in an external editor using the "Edit" command.

4.3 Configuring screen objects

4.3.1 Select multiple objects

Introduction

To align the object with one another or rotate them, select all affected objects. This procedure is called "multiple selection."

The Inspector window shows all the properties of the selected objects.

You have the following options to select multiple objects:

- Draw a selection border around the objects.
- Hold down the <Shift> key, and click the required objects.

Selection border of a multiple selection

The selection border surrounds all objects of a multiple selection. The selection border is comparable with the bounding box that surrounds an object.

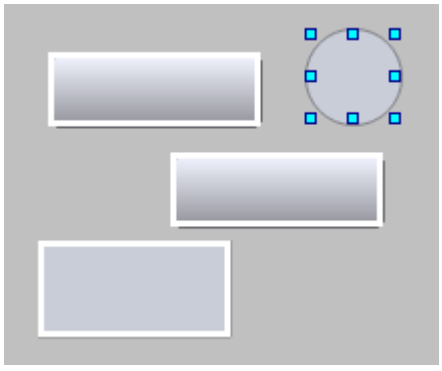
The selection border is only visible as long as it is pulled up with the mouse button pressed. When you have made your multiple selection, the following border is displayed:

- The reference object is indicated by the bounding box.
- The other selected objects are indicated by a border. The color of the border depends on the background color of the screen.

The color of the selection border is adapted to the background color of the screen in such a way that the selection border is correctly displayed and visible with multiple selection.

Specifying a reference object

The reference object is the object upon which the other objects are oriented. The reference object is framed by a bounding box with handles. The following figure shows a reference object with three additional selected objects:



You have the following options to specify the reference object:

- Select the objects via multiple selection. The object selected first is then the reference object.
- Draw a selection border around the objects. As a reference object, the object is automatically defined as on top in the foreground. If you wish to specify a different object within the selection as the reference object, click on the desired object. This action does not cancel your multiple selection.

Requirement

- The HMI screen is open with at least two objects.

Selecting multiple objects with a selection border

To select multiple objects using a selection border, follow these steps:

1. Position the mouse pointer in the work area close to one of the objects to be selected.
2. Hold down the mouse button, and draw a selection border around the objects to be selected.

Selecting multiple objects using the <Shift> key

To move multiple objects with the <Shift> key, follow these steps:

1. Hold down the <Shift> key.
2. Click the relevant objects, working in succession.
   All the selected objects are identified by borders.
   The object selected first is identified as reference object.

Note

To remove an object from the multiple selection, press <Shift>, hold it down and then click the relevant object once again.

Result

Multiple objects are selected. An object is identified as the reference object. You can perform the following steps:

- Move or rotate all objects together.
- To resize all objects by the same ratio, drag the selection border to increase or reduce the size.
- Align the objects to the reference object.
- Change object properties.

4.3.2 Copying objects

Introduction

You can copy objects individually or with a multiple selection.

Requirement

- The HMI screen is open with at least one object.

Copying objects

To copy an object, follow these steps:

1. Select the object.
2. Copy the object with <Ctrl + C> or select "Copy" in the shortcut menu.
3. Paste the object with <Ctrl + V> or select "Paste" in the shortcut menu.
4. Drag and drop the copied object to the required position.

If you want to copy multiple objects using multiple selection, draw a selection frame around the desired objects and proceed as described above.

Using copied objects with the same spacing

If you want to use the copied object multiple times, proceed as described above. Press <Ctrl + V> or select "Paste" repeatedly to automatically paste the objects in the HMI screen with equal spacing.

The same principle also applies to multiple selection of the objects.

4.3.3 Creating objects automatically

Introduction

You can automatically create the objects by dragging a screen, a graphic, a graphic list, a text list or a tag into an HMI screen.

Requirement

- The HMI screen is open.

Creating objects automatically

You can create the following objects automatically:

- Button: Drag an HMI screen from the project tree to another screen.
- Graphic view:
  - Drag a graphic from the detail view of the project graphics into the screen.
  - Drag a graphic list from the detail view of the text and graphic lists into the screen.
- Text box: Drag a text list from the detail view of the text and graphic lists into the screen.
- IO field: Drag a tag from the detail view of the tag table into the screen.

4.3.4 Defining the output format

Introduction

In many objects you can adjust the output format for the displayed values or define it yourself. You can process and output the process value that is displayed in the object in different notations. You can select frequently used output formats directly in the user interface. You can adjust the formatting codes or define them yourself.
You can define the output of a screen object in "Properties > General > Output format" for the following data:

- Floating-point numbers
- Binary
- Hexadecimal
- Decimal
- Text
- Duration, date, time
- Percent, currency, unit
- Numerical values

The definition of the output format is based on UNICODE CLDR. You can find additional information on the CLDR project and on the definitions on the Internet at http://cldr.unicode.org/ (http://cldr.unicode.org/)

Requirement

- The HMI screen is open with at least one object.

Defining the output format

You can define the output format by stringing together formatting codes. The formatting codes act as placeholders for a specific group of characters. If, for example, a formatting code that only allows the display of the digits 0-9 is specified for a position in the display of the IO field, you cannot input letters at this position.

The definitions for the output format are independent of the language. The output format can be language-specific and thus take linguistic differences into account, for example, for output of the date.

You can define and combine different format patterns yourself.

	**Efficiency tip**
• Select the output format from the drop-down list or edit it manually in the input window. You can also customize the combined output formats, for example, change the output format "{D} {T}" manually to "{D,long} {T}".	

The tables below show examples for the definition of output formats that are frequently used.

"Binary" data format

You use the "Binary" data format to display binary values. The "Binary" data format has the following inputs:

- Sign "B"
- Number of digits (optional)
- Information on forming blocks (optional)

Output format example	Mindigits	Block size	Tag value	Result
{B}	Default	-	16	1 0000
{B8}	8	-	16	0001 0000
{B8}	8	-	80	0101 0000
{B8,4}	8	4	80	0101 0000
{B,2}	Default	2	80	1 01 00 00
{B}	Default	-	-1	1111 1111

Mindigits	Number of digits (optional)	Minimum: 1	Maximum: 64	Default value: 1
Block size	Number of digits in front of the separator (optional)	Minimum: 0 (none)	Maximum: 8	Default value: 4

Note

The "Binary" data format does not support any negative values in Unified Runtime.

If you wish to output negative values, use the "Integer" or "Float" data formats with the {F} or {N} sign.

"Hexadecimal" data format

You use the "Hexadecimal" data format to display hexadecimal values. The "Hexadecimal" data format has the following inputs:

- Sign "H"
- Number of digits (optional)
- Information on forming blocks (optional)

Output format example	Mindigits	Block size	Tag value	Result
{H}	Default	-	1	1
{H}	Default	-	15	F
{H}	Default	-	45054	AFFE
{H4,2}	4	2	45054	AF FE
{H,2}	Default	2	45054	AF FE

Mindigits	Number of digits (optional)	Minimum: 1	Maximum: 16	Default value: 1
Block size	Number of digits in front of the separator (optional)	Minimum: 0 (none)	Maximum: 8	Default value: 4

Note

The "hexadecimal" data format does not support any negative values in Unified Runtime.

If you wish to output negative values, use the "Integer" or "Float" data formats with the {F} or {N} sign.

"Integer" data format

You use the "Integer" data format to display decimal values. The "Integer" data format has the following inputs:

- Sign "I"
- Number of digits (optional)
- Plus or minus sign in front of the sign (optional)

Output format example	Mindigits	Tag value	Result
{I}	Default	9	9
{I4}	4	9	0009
{0000}	Default	9	0009
{I2}	2	123	123
{I}	Default	1.6	1
{+I}	Default	1	+1
{I1}	1	123456789	123456789
{#,##0}	Default	1,234	1,234
{E}	Default	1.12E+3	1.12E+3
{E3}	3	1.123E+3	1.123E+3

+/-	Sign (optional)			Default value: None
Mindigits	Number of digits (optional)	Minimum: 1	Maximum: 16	Default value: 1

"Float" data format

You use the "Float" data format to display values with floating-point numbers. The "Float" data format has the following inputs:

- Sign "F", "N", "E"
- Number of decimal places (optional)
- Plus or minus sign in front of the sign (optional)

Output format example	Mindigits	Tag value	Result
{F}	default	123.456	123.45
{+F}	default	123.1	+123.10
{F3}	3	123.123	123.123
{N}	default	123	123.00
{+N}	default	123	+123.00
{N1}	1	123	123.0
{#,##0.###}	default	1234.567	1,234.567
{#,##0.##}	default	1234.123	1234.12
{#,###.#}	default	1234.123	1,234.1
{E}	default	1123	1.12E+3
{E1}	1	1123	1.1E+3
{E3}	3	1123	1.123E+3
{+E}	default	1123	+1.12E+3
{E0}	0	1123	1E+3

+/-	Sign (optional)			Default value: None
Mindigits	Number of decimal places (optional)	Minimum: 1	Maximum: 16	Default value: 2

"String" data format

You use the "String" format to display texts. The "String" data format has the following inputs:

- Sign "S"
- Number of characters (optional)
- Formatting parameters (optional)

Output format example	Maxchars	String format	Tag value	Result
{S}	Default	-	Motor	Motor
{S4}	4	-	Motor	Moto
{S,trim}	Default	trim	Motor	Motor
{S,upper}	Default	upper	Motor	MOTOR
{S,lower}	Default	lower	Motor	motor
{S,trim,upper}	Default	trim, upper	Motor	MOTOR
{S3,trim,upper}	3	trim, upper	Motor	MOT

Maxchars	Number of characters (optional)	Minimum: 1	Maximum: 99	Default value: Complete input
String format	Parameters for formatting of the input (optional)	trim: Outputs the input string without spaces. upper: Outputs the input string in uppercase letters. lower: Outputs the input string in lowercase letters.		

"Duration" data format

The accuracy of the duration inputs is limited to 1 ms. All inputs of less than 1 ms are shown as 0 in runtime.

To display fractions of a second, use .S, .SS or .SSS according to the pattern for the duration.

The "Period" data format has the following inputs:

- Sign "P"
- Number of time units (optional)

Output format example	Tag value (ns)	Result
{P}	1:20	1:20
{P,s}	10000000	1
{P,s}	-10000000	-1

Output format example	Tag value (ns)	Result
{P,s}	10000000	01
{P,m:ss}	35990000000	59:59
{P,h:mm:ss}	36000000000	1:00:00
{P,hh:mm:ss}	36000000000	01:00:00
{P,D hh:mm:ss}	864000000000	1D 00:00:00
{P,DD hh:mm:ss}	864000000000	01D 00:00:00
{P,s.S}	10000	0.0
{P,s.SS}	10000	0.00
{P,s.SSS}	10000	0.001
{P,s.SSS}	9999	0.000

Durationunit Duration

The output format {P} enables automatic mode. In the mode the result with the smallest necessary unit of time is written. The table below shows some examples for tag values and their output in automatic mode.

Tag value (ns)	Result	Meaning
9999	0	0.9999 ms
10000	0.001	1ms
9990000	0.999	999ms
10000000	1	1s
10010000	1.001	1s 1ms
600000000	1:00	1m
700000000	1:10	1m 10s
35999990000	59:59.999	59m 59s 999ms
36000010000	1:00:00.001	1h 1ms
863999990000	23:59:59.999	23h 59m 59s 999ms
937845670000	1D 02:03:04.567	1D 2h 3m 4s 567ms
86400000000000	100D	100D

Localized output: "Date" and "time" data format

"Date" and "time" data formats can be localized. The output format depends on the system language. The table below shows the examples for "German".

Data format	Permitted values and default values	Output format example	Tag value	Result
Date: {D,Length}	Length: short medium long or code according to CLDR format with the sign @ Default value: short	{D}	2019-03-21T21:08:33	21.03.19
		{D,medium}	2019-03-21T21:08:33	21.03.2019
		{D,long}	2019-03-21T21:08:33	March 21, 2019
		{D,@y}	2019-03-21T21:08:33	2019
		{D,@dd.MM.yyyy}	2019-03-21T21:08:33	21.03.2019
		{D,@dd. MMMM yyyy}	2019-03-21T21:08:33	March 21, 2019
		{D,@dd.MM}	2019-03-21T21:08:33	21.03
		{D,@dd. MMM}	2019-03-21T21:08:33	March 21
		{D,@MM-dd-yyyy}	2019-03-21T21:08:33	03-21-2019
		{D,@MMM dd, yyyy}	2019-03-21T21:08:33	March 21, 2019
		{D,@M.d}	2019-03-21T21:08:33	3.21
		{D,@MMMM dd}	2019-03-21T21:08:33	March 21
		{D,@yyyy/mm/dd}	2019-03-21T21:08:33	2019/03/21
		{D,@EEE, MMM dd, 'yy}	2019-03-21T21:08:33	Thu., June 15, '19
		{D,@EEEE, MMMM d, yy}	2019-03-21	Thursday, March 21, 19
Time: {T,Time format}	Time format: short medium medium.S medium.SS medium.SSS or code according to CLDR format with the sign @ Default value: medium	{T}	2019-03-21T21:08:33	21:08:33
		{T,short}	2019-03-21T21:08:33	21:08
		{T,@h:mm a}	21:08:33	9:08 p.m.
		{T,@HH:mm}	21:08:33	21:08
		{T,@hh:mm a}	21:08:33	9:08 p.m.
		{T,@HH:mm:ss}	21:08:33	21:08:33
		{T,@HH:mm:ss.SSS}	21:08:33.1230	21:08:33.123
		{T,medium.SSS}	21:08:33.1239	21:08:33.123

Combined output

You can combine the output formats:

Description	Output format example	Tag value	Result
Date and time	{D} – {T}	2019-03-21T21:08:33	21.03.2019 – 21:08:33
	{D} {T}	2019-03-21T21:08:33	21.03.19 21:08:33
	{D,@EEEE, dd. MMMM yyyy, h:mm:ss a}	2019-03-21T21:08:33	Donnerstag, 21. März 2019, 9:08:33 nachm.
	{D,@dd.MM.yyyy HH:mm}	2019-03-21T21:08:33	21.03.2019 21:08
Date and time with line break	{D}\n{T}	2019-03-21T21:08:33	21.03.2019 21:08:33
Two numerical values	hex {H} – dec {I}	45054	hex AFFE – dec 45054
Text with prefix	myMotor {S}	motor34	myMotor motor34
Number with prefix	MyMotor#{00}	12	MyMotor#12
Percent	{I}%	20	20%
Currency	{#,##0} EUR	20	20 EUR
Currency	${#,##0.##}	20	$20
Unit	{F2} m/s	10.00	10.00 m/s

See also

Configuring an alarm control (Page 753)

http://cldr.unicode.org/ (http://cldr.unicode.org/)

4.3.5 Disable remote control

Introduction

You can disable the "Allow operator control" property for operable objects. You cannot then operate the objects in runtime. The object or the buttons on the object are dimmed.

Requirement

- The HMI screen is open with at least one object.

Disable the "Allow operator control" property

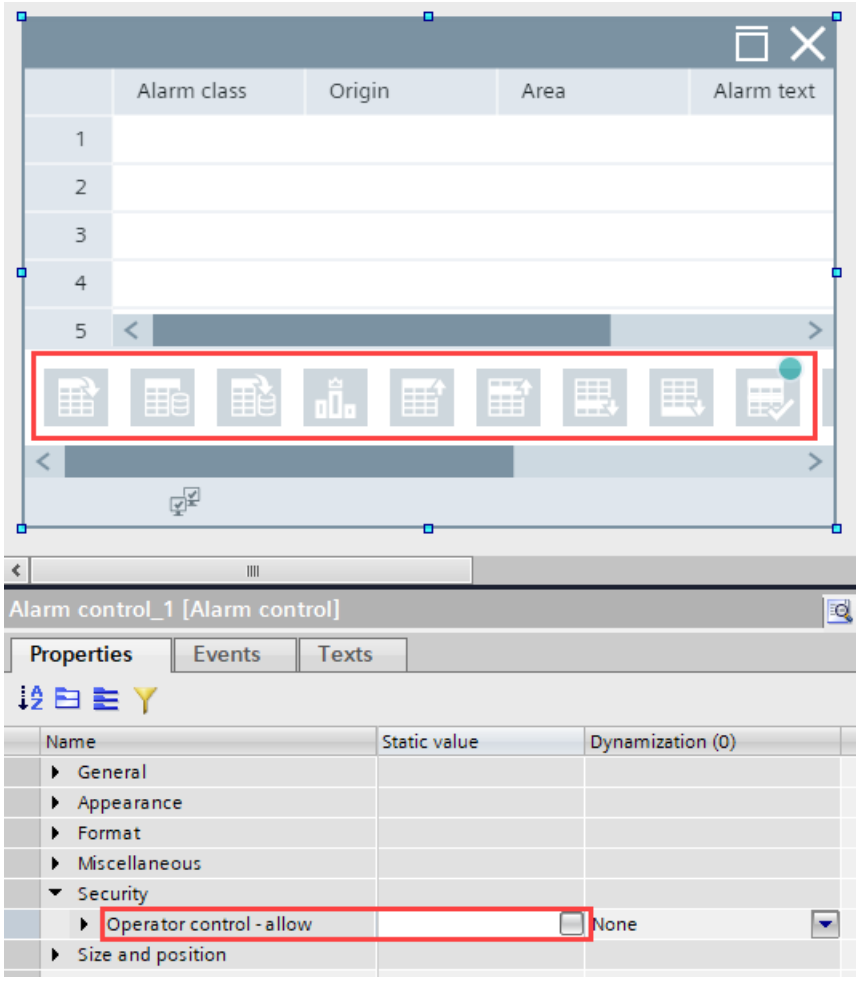
You can disable the option for operator control in runtime using the "Allow operator control" property.

The disabled property "Allow operator control" has the following effect:

- The objects in the "Elements" group are grayed out.
- For objects in the "Controls" group, the operable areas, e.g. buttons on the toolbar, are grayed out.

The objects are displayed in the same way in the engineering system and in runtime.

The following figure shows the grayed out buttons in the alarm control:



4.3.6 Hotkeys

Introduction

You can use the hotkeys in Runtime to perform various actions. You can specify the hotkeys as a key or key combination in the Engineering System.

Examples of hotkey functions:

- Change language
- Change screen
- Acknowledge alarm

The use of hotkeys is supported for WinCC Unified PCs.

The use of hotkeys is not supported for Unified Comfort Panels.

Note

Hotkeys are also supported for the View Of Things application.

Hotkeys are not supported for faceplates.

Note

It is recommended to use the US English layout keyboard.

Requirement

- The HMI screen is open with at least one button or a Control.

Supported objects

You can configure hotkeys in the "Properties" tab of the Inspector window for the following objects:

- Button
- Controls with one toolbar

The default value for a hotkey in the "Static value" column is "None".

Configuring hotkeys

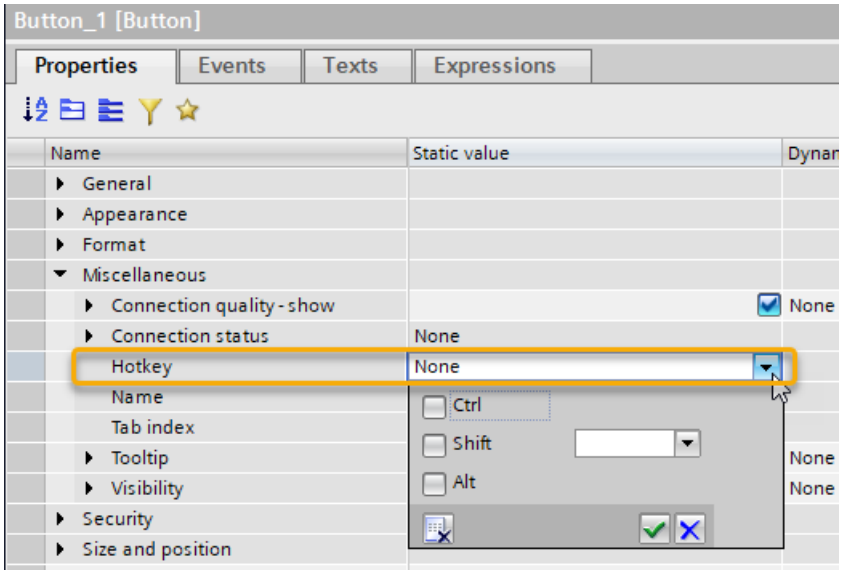
You can find the "Hotkey" property in the Inspector window in the "Properties" tab:

- Button: "Properties > Miscellaneous > Hotkey".
- Controls with one toolbar: "Properties > Miscellaneous > Toolbar > Elements > [0] Button > Hotkey".

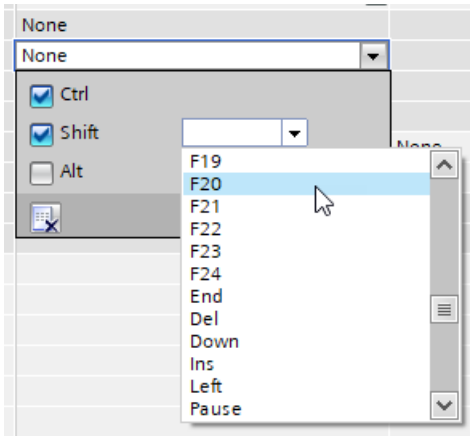
To configure a hotkey for a button, for example, follow these steps:

1. Select the button.
2. Select "Properties > Miscellaneous > Hotkey".

3. In the input field, click in the "Static value" column.
   A dialog opens.



4. Select a keyboard shortcut.



Note

You may use a keyboard shortcut only once in a screen. In case of duplicate use, an error message will appear.

5. Using the options in the selection dialog, you can do the following:
   - Delete settings and close the dialog .
   - Discard changes and close the dialog .
   - Apply settings and close the dialog .

Result

You have configured a key or key combination for a hotkey.

4.3.7 Configuring object properties

4.3.7.1 Managing object properties

Introduction

The properties of an object are displayed in the property list in the Inspector window. Here you can edit the properties, e.g. change the size and position of an object, or dynamize objects.

You manage the properties via the  icons.

Display of the property list in the Inspector window

The properties are displayed in the property list either in alphabetical order or in categories.

You can sort the property list as follows:

- Display of properties in alphabetical order
- Display of the properties grouped in categories

With both views, all details of the individual properties can be shown or hidden:

- All details are shown
- All details are hidden

	**Tip for working effectively**
• Drag-and-drop a tag or resource list from the details view in the project tree to a property in the Inspector window. The details of the lower-level of the property are displayed.	

"Filter" function

You can locate the individual properties by using the "Filter" function .

"Favorites" function

You define the favorite object properties via the function "Favorites" .

Dynamized properties

Dynamized properties are shown in dark blue font and in bold in the Inspector window in the "Name" column.

Groups containing a dynamized property are shown in dark blue font.

See also

Displaying dynamization of the properties (Page 463)

"Filter" function (Page 376)

Adding an object property to favorites (Page 377)

4.3.7.2 "Filter" function

Introduction

You can locate the individual properties of an object by using the "Filter" function in the Inspector window.

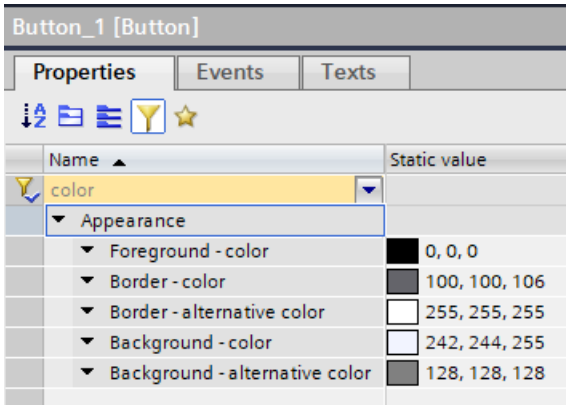
Requirement

- The HMI screen is open with at least one object.

Filter properties

To filter the properties of a screen object, follow these steps:

1. Click on the "Filter" icon in "Properties > Properties" in the Inspector window.
2. In the "Search" input field, type in the term you are looking for, e.g. "Color" in the "Name" column.
3. Confirm the input with the <Return> key.

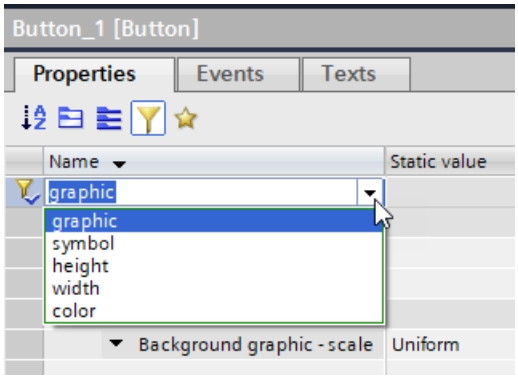


All hits of the properties which contain the term "color" are displayed.

Using search terms with other objects or screens

If you choose another screen object or another screen, the current search term remains in the input field.

The search terms that you have entered in the currently open project are retained until the opened project is closed.



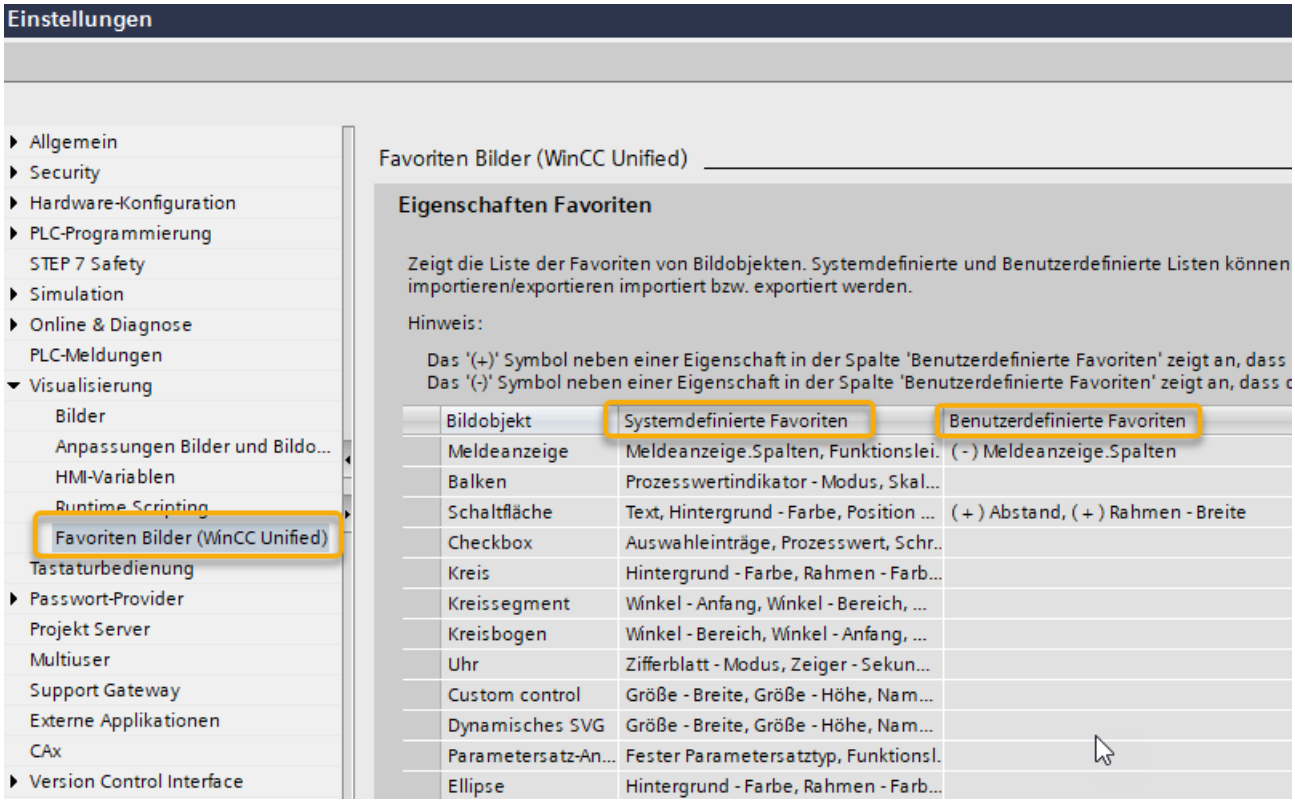
4.3.7.3 Adding an object property to favorites

Introduction

You can define your own favorite properties for each screen object.

Some properties are defined as favorites by the system.

You can find an overview of system-defined favorites under "Settings > Visualization > Favorites screens (WinCC Unified)" in the "Favorites properties" table:



The user-defined favorites are shown in the "User-defined favorites" column in the table.

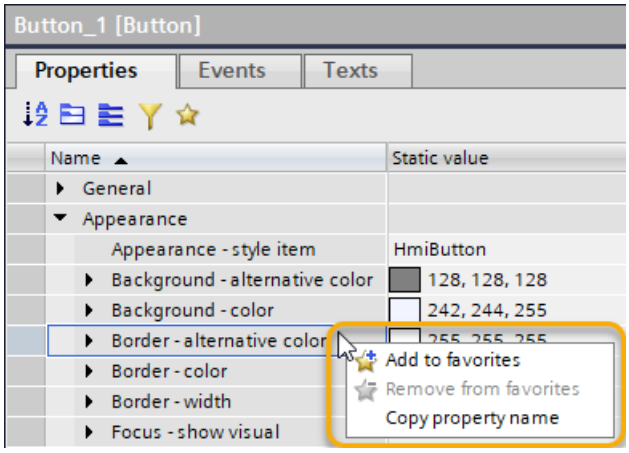
You can also locate the individual properties of an object by using the "Filter" function in the Inspector window .

Add property to favorites

You can add your favorite properties of a screen object to the favorites.

To add a property to the favorites, follow these steps:

1. Right-click on a property that is not defined as a favorite by the system.
   The shortcut menu opens.

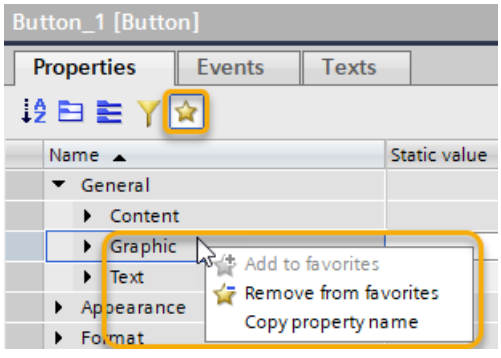


2. Select "Add to favorites". The number of favorite properties is not limited.
3. To display all favorites, click the icon .

Remove property from favorites

To remove a property from the favorites, follow these steps:

1. To display all favorites, click the icon .
2. Right-click on the favorite property.
   The shortcut menu opens.



3. Select "Remove from favorites".

You can find all favorites added or deleted by you in the "Favorite properties" table under "Settings > Visualization > Favorites screens (WinCC Unified)".

See also

Managing object properties (Page 375)

4.3.7.4 Changing a property for multiple objects

Introduction

You can change the static value of a property for multiple objects at the same time.

Requirement

- The HMI screen with at least two objects is open.

Changing properties for multiple objects

To change the static value of a property for multiple objects at the same time, follow these steps:

1. Select several objects in the screen via multiple selection.
2. Select a property, e.g. the background color, in the Inspector window.
3. Change the static value of the property.

The property is changed on all selected objects that have this property.

See also

Select multiple objects (Page 361)

4.3.7.5 Automatically filling in of property values for an object collection

Introduction

You can automatically fill in the values of properties of an object collection.
Examples of object collections are:

- Points for:
  - Polygon
  - Polyline
- Entries for:
  - Check box
  - Radio button
  - List box
- Columns of an alarm control

- Trend areas of a trend control
- Symbols of a toolbar of a screen object

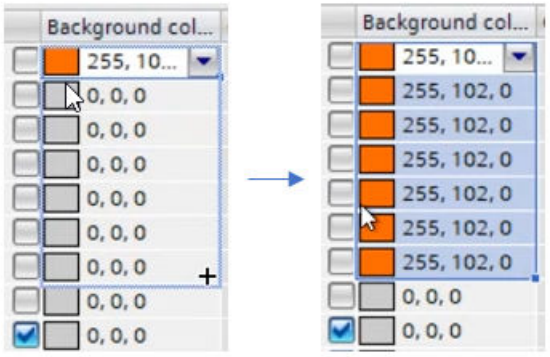
Requirement

- The HMI screen is open with at least one object.

Automatically fill in property values for objects

To automatically fill in the values of properties for objects, follow these steps:

1. Select the collection according to the object:
   - For polygons and polylines, click "Properties > Properties > Size and position > Points".
   - For the elements, e.g. check box, click "Properties > Properties > General > Selection items".
   - With the controls, e.g. alarm control, click "Properties > Properties > Miscellaneous > Alarm control > Columns".
   - For the controls with a toolbar, click "Properties > Miscellaneous> Information bar > Elements".
2. Select one or more contiguous cells in the collection in the right part of the Inspector window.
3. Drag the blue border around this cell up or down.



The values are transferred to the destination cells.

Moving and deleting cells

To move or delete the cells, follow these steps:

1. In the right part of the Inspector window, drag the blue border around the cells.
2. Move or delete the selected cells using the ✕ and ↥ ↧ buttons.

The cells are moved or deleted.

See also

Polyline (Page 277)

Polygon (Page 278)

Dynamizing an object property with a tag (Page 467)

4.3.8 Designing objects

4.3.8.1 Changing the object size

Introduction

When you select an object, it is framed by a bounding box with blue handles. You have the following options for resizing an object:

- Use the mouse to drag the blue handles on the bounding box.
- Configure the properties in the Inspector window.

Note

You can change the form of the objects "Line", "Polyline" and "Polygon" as follows:

- Use the mouse to drag the orange handles on the object.
- Configure the properties in the Inspector window.

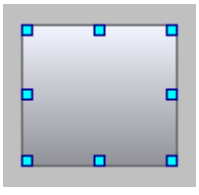
Requirement

- The HMI screen is open with at least one object.

Changing object size with the mouse

To change the object size with the mouse, follow these steps:

1. Select the object you want to resize.
   The bounding box is displayed. The following figure shows a selected object:



2. Drag a handle of the box to a new position.

The object size is changed.

	**Tips for working effectively**
If you press the <Ctrl + Shift> keys while dragging, the object size is changed according to the aspect ratio.	

Changing the size of multiple objects with the mouse

To change the size of multiple objects with the mouse, follow these steps:

1. Select the objects by multiple selection.
   The bounding box is displayed.
2. Drag a handle of the box to a new position.

The size of the selected objects is changed.

	**Tips for working effectively**
If you press the <Ctrl + Shift> keys while dragging, the selected objects are resized according to the aspect ratio.	

Configuring the object size through properties

To change the object size through properties, follow these steps:

1. Select "Properties" > "Properties" > "Size and position".
2. In the "Static value" column, enter the "Size - width" and "Size - height" coordinates.

The object size is changed.

Changing object size using keys

To change the object size using keys, follow these steps:

1. Select the object you want to resize.
2. On the keyboard, press the keys:
   - <Ctrl + Arrow key>
   - <Ctrl + Shift + Arrow key>

The object size is changed depending on the arrow key selected.

4.3.8.2 Changing the position of an object

Introduction

When you select an object, it is framed by a bounding box with blue handles. You have the following options for repositioning an object:

- Position the object with the mouse.
- Configure the object properties in the Inspector window.

Requirement

- The HMI screen is open with at least one object.

Changing the object position with the mouse

To change the object position with the mouse, follow these steps:

1. Select the object whose position you want to change.
   The bounding box is displayed.
2. Left-click the object and keep the left mouse button pressed.
3. Move the object with the mouse pointer to the new position.

The object is moved to the new position.

Configuring the object position through properties

To change the object position using properties, follow these steps:

1. Select "Properties" > "Properties" > "Size and position".
2. In the "Static value" column, enter the "Position - left" and "Position - top" coordinates.

The position of the object in reference to the screen origin is changed. The zero position is located at the top left-hand corner of the screen.

Changing the object position using keys

To change the object position using keys, follow these steps:

1. Select the object you want to resize.
2. On the keyboard, press the <Shift + Arrow> keys

The object position is changed depending on the arrow key selected.

4.3.8.3 Transfer format

Introduction

You can transfer the format of an object to another object. You can find the "Transfer format" icon in the function bar of the screen editor.

Transferring the format

You use the "Transfer format" function to transfer the properties of the source object to the target object.

You can transfer edit the properties:

- From a source object to the same target object, for example, from line to line.
- From a source object to a different target object, for example, from line to a circle.
- From a source object to multiple target objects, for example, from line to line, circle and button.

Requirement

- The HMI screen is open with at least one object.

Transferring the format

To transfer the properties of an object to a single target object, follow these steps:

1. Select an object.
2. Click on the "Transfer format" icon. The mouse pointer icon changes.
3. Click on another object.

The properties are transferred to the target object.

To transfer the properties of an object to multiple target objects, follow these steps:

1. Select an object.
2. Click on the "Transfer format" icon. The mouse pointer icon changes.
3. Click on one object after another.

The properties are transferred to multiple target objects.

Table of objects and categories of the properties

The table below contains the objects and categories of the properties that you can transfer to another object.

These properties can be found in the Inspector window under:

- "Properties > General"
- "Properties > Format"
- "Properties > Appearance"

The properties that you cannot transfer to another object are listed in the "With the exception of" column.

Object	All properties of the category	With the exception of
Line, polyline, polygon, ellipse, ellipse segment, circle segment, elliptical arc, circular arc, circle, rectangle	Appearance	
Text box	General	Text
	Format	
	Appearance	
Graphic view	Format	
	Appearance	
IO field	General	Output format Mode Process value
	Format	
	Appearance	
Symbolic IO field	General	Process value Resource list
	Format	Content > Type
	Appearance	
Button, switch	General	Content > Type
	Format	
	Appearance	
Bar	General	Label > Text Scale > Scale value - maximum, Scale value - minimum Process value Title > Text
	Appearance	
Slider	General	Label > Text Title > Text
	Appearance	
Gauge	General	Label > Text Title > Text Process value Scale > Output format Scale > Scale value - maximum, Scale value - minimum
	Appearance	

Object	All properties of the category	With the exception of
Check box, radio button, list box	Format	Content > Type
	Appearance	
Clock	General	
	Appearance	
	Miscellaneous	Name Time - source Tab index Title > Text Tooltip Connection quality - show Connection status
Touch area	Appearance	
Alarm control	Format	
	Appearance	
	Miscellaneous	Alarms - show current Alarms - current Alarms - displayed Label > Text Alarm control > Header - settings, Color mode, Filter - allow, Selection - select entire rows, Selection - mode, Sorting - allow, Columns, Row height Alarm statistics - view > Header - settings, Color mode, Filter - allow, Selection - select entire rows, Selection - mode, Sorting - allow, Columns, Row height Alarm statistics - settings Name Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Time zone

Object	All properties of the category	With the exception of
Media Player	Appearance	
	Miscellaneous	Autoplay Label > Text Name Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Video output
Screen window	Format	
	Appearance	
	Miscellaneous	Label > Text Screen name Screen number Tab order - Continue in screen window Name Icon System Tab index Connection status
Trend control	Appearance	
	Miscellaneous	Label > Text Name Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Time zone

Object	All properties of the category	With the exception of
Trend companion	Format	
	Appearance	
	Miscellaneous	Label > Text Appearance - trend ruler > Header - settings, Color mode, Filter - allow, Selection - select entire rows, Selection - mode, Sorting - allow, Columns, Row height Name Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Time zone
Process control	Appearance	
	Miscellaneous	Editing mode Label > Text Name Online Process control > Header - settings, Color mode, Selection - select entire rows, Selection - mode, Sorting - allow, Columns, Row height Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Time factor - average Time zone

Object	All properties of the category	With the exception of
Function trend control	Appearance	
	Miscellaneous	Axes - swap Label > Text Name Online Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status
Web control	Appearance	
	Miscellaneous	Label > Text Name Online Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status
Parameter set control	Appearance	
	Miscellaneous	Editing mode Label > Text Details - hide Name Parameter view > Header - settings, Color mode, Filter - allow, Selection - select entire rows, Selection - mode, Sorting - allow, Columns, Row height Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Time zone

Object	All properties of the category	With the exception of
Faceplate container	Appearance	
	Miscellaneous	Label > Text Faceplate type Name Interface Icon Tab index Connection status
System diagnostics control	Appearance	
	Miscellaneous	Label > Text Name Information bar > Operator control - allow, Elements, Tooltips - show Icon Function bar > Operator control - allow, Elements, Tooltips - show Tab index Connection status Time zone

4.3.8.4 Designing the fill pattern

Introduction

You can design the fill pattern of an object. The design options change in the Inspector window depending on the object for which you are making the filling pattern.

For certain objects, you can define a color background, a transparent background or a background with a color gradient.

Requirement

- The HMI screen is open with at least one object.

Designing the fill pattern of an object

To design the fill pattern of an object, e.g. a circle, follow these steps:

1. In the Inspector window, click "Properties > Properties > Appearance > Background - Fill pattern".
2. To define a transparent background for the object, for example, select "Transparent".

	**Efficiency tip**
Using multiple selection, you can set the fill pattern in multiple objects at the same time.	

Result

The object is shown as transparent.

Additional design options

Additional design options are available in the Inspector window under "Properties > Properties > Appearance". The procedure for using these options is the same as the one described in the examples above.

Restriction for objects with events

Events for operator actions are only triggered if the operator action takes place in the marked, visible area of the object.

If the fill pattern of an object is transparent, click exactly on the object border in runtime in order to trigger the events configured for the object. Select the border width so that you can hit the border.

Note

Objects for which the "Opacity" property has the value "0" are also not visible in runtime and do not trigger events.

4.3.8.5 Designing the border of an object

Introduction

Elements and some basic objects have a border. You can configure the width and color of the border.

Requirement

- The HMI screen is open.
- An element or one of the following basic objects is placed:
  - Text box
  - Rectangle
  - Circle and circle segment
  - Ellipse and ellipse segment
  - Polyline

Change frame width

1. Select the object.
2. In the Inspector window, click "Properties > Properties > Appearance > Border - width".
3. Enter the desired width.

For elements and text boxes, the border is drawn on the inner edge of the object; for the other basic objects, the border is drawn on the inner and outer edges. The values of the "Size - width" and "Size - height" properties remain the same.

Change frame color

1. Select the object.
2. Click "Properties > Properties > Border - color" in the Inspector window.
3. Enter the RGB values of the color or select a color from the drop-down list.
   Select "More colors" in the drop-down list to add custom colors to the selection, specify HSL, and more.

Note

Alternatively, you can change the color in the central color management of the object.

See also

Central color management (Page 401)

4.3.8.6 Configuring reordering of the columns

Introduction

You have the option of configuring the following table-based controls in such a way that operators can rearrange the table columns in runtime:

- Alarm control
- Trend companion
- Parameter set control
- System diagnostics control
- Process control

Requirement

- The screen editor is open.
- One of the above-named controls has been placed on the screen.

Procedure

1. Select, for example, an alarm control in the work area.
2. In the Inspector window, click "Properties > Properties > Miscellaneous > Alarm control > Header - settings".
3. To allow the columns to be rearranged in runtime, enable the property "Columns - Change sequence".
   To prevent reordering, deselect the property.

Result

Once the project has been downloaded to the device, operators can use drag-and-drop to reorder the columns displayed in the control in runtime.

4.3.8.7 Rearranging columns in runtime

Introduction

You can reorder the table columns in the following table-based controls:

- Alarm control
- Trend companion
- Process control
- Parameter set control
- System diagnostics control

Requirement

Reordering of columns is allowed by the configuration of the control in the engineering.

Procedure

Using drag-and-drop, move the header of the column to be moved onto the header of another column.

Note

The time column cannot be moved.

Result

The moved column is inserted at the position that the mouse pointer had when the drag-and-drop movement ended.

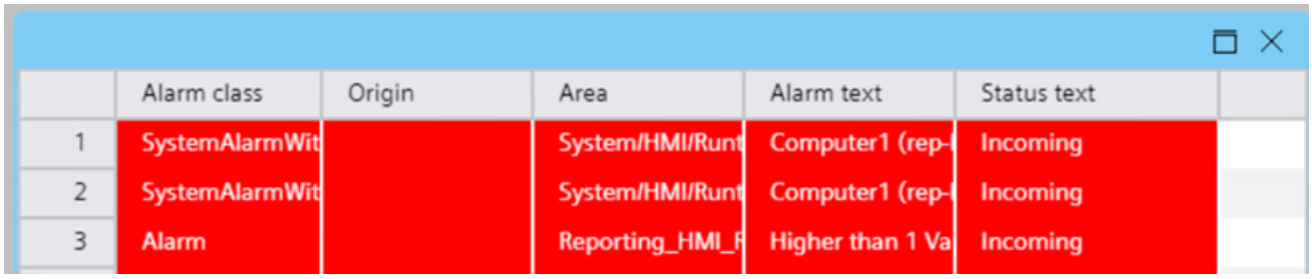
The new column order applies only to the current client. If you switch to another page or refresh the browser window, the column order is lost.

Note

If you move a column next to a hidden column and then show this column afterwards, it is always displayed to the right of the moved column.

Example 1: Inserting columns to the right or the left

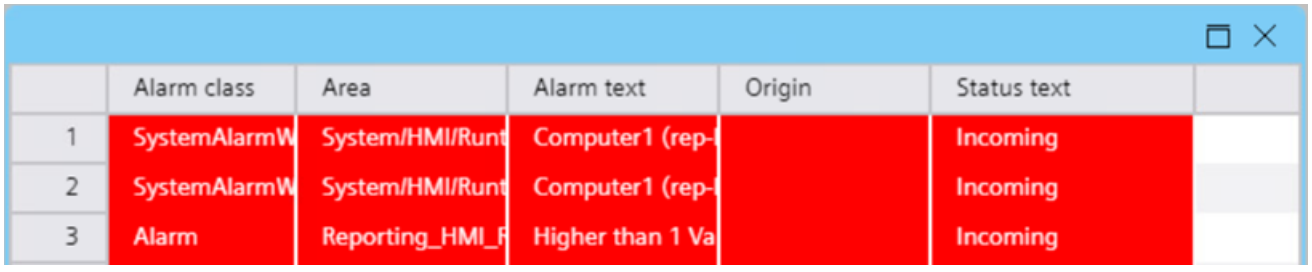
The procedure is illustrated using the example of an alarm control. In the initial situation, the alarm control table has the following column arrangement:



To insert the "Origin" column to the right of the "Alarm text" column, follow these steps:

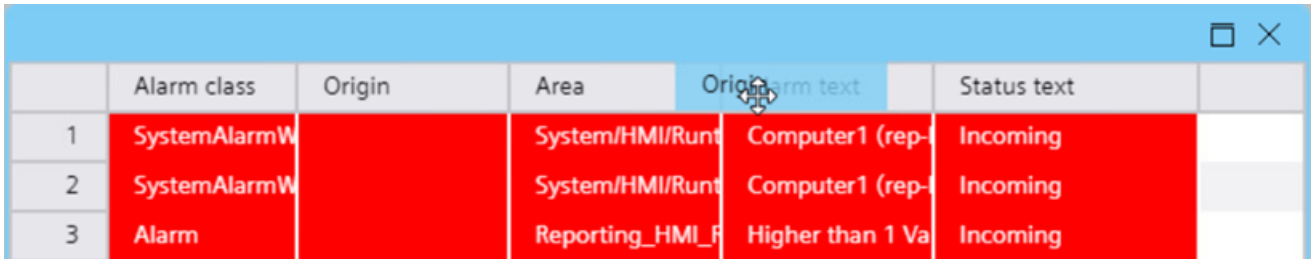
1. Drag-and-drop the column header from "Origin" to the right column header half of the "Alarm text" column:

2. The "Origin" column is inserted to the right of the "Alarm text" column.



To insert the "Origin" column to the left of the "Alarm text" column, follow these steps:

1. Drag-and-drop the column header from "Origin" to the left column header half of the "Alarm text" column:



2. The "Origin" column is inserted to the left of the "Alarm text" column.

Example 2: Reordering of columns in combination with hidden columns

This example illustrates the reordering of columns in combination with hidden columns.

- The alarm control has the same column order as in Example 1.
- The alarm control has been configured in the engineering system in such a way that the display of the "Origin" column is controlled dynamically in runtime by setting a tag.

To reorder the columns in combination with hidden columns, follow these steps:

1. Hide the "Origin" column by setting the tag.
2. Insert the "Status text" column to the left of the "Area" column.
3. Show the "Origin" column by setting the tag.

The order of the columns is as follows: "Alarm class", "Status text", "Origin", "Area", "Alarm text".

4.3.9 Moving objects

4.3.9.1 Aligning objects

Introduction

You can align the screen objects in the screen with reference to a reference object.

Requirement

- The HMI screen is open with at least two objects.

Aligning objects flush

The selected objects will be aligned flush to the reference object. The reference object is the object that you selected first.

Icon	Description
	Aligns the selected objects to the left edge of the reference object.
	Aligns the selected objects to the vertical center axis of the reference object.
	Aligns the selected objects to the right edge of the reference object.
	Aligns the selected objects to the upper edge of the reference object.
	Aligns the selected objects to the horizontal center axis of the reference object.
	Aligns the selected objects to the lower edge of the reference object.
	Centers the selected objects to the center points of the reference object.
	Centers the selected objects vertically in the screen.
	Centers the selected objects horizontally in the screen.
	Distributes at least 3 selected objects evenly horizontally.
	Distributes at least 3 selected objects evenly vertically.
	Specifies the same height for all selected objects.
	Specifies the same width for all selected objects.
	Specifies the same height and width for all selected objects.

Aligning selected objects

To align the selected objects in relation to a reference object, follow these steps:

1. Select the desired objects using multiple selection.
2. Specify an object as the reference object.
3. Select the desired command for alignment in the toolbar or the shortcut menu.

The selected objects are now aligned in relation to a reference object.

Aligning individual objects

To align an object in the screen in relation to other objects using minor lines, follow these steps:

1. Select an object.
2. Drag the object to a position until the blue minor lines become visible.
3. Position the object using the auxiliary lines.

The object is now aligned in relation to other objects.

4.3.9.2 Move objects

Introduction

There are various ways in which you can move objects individually or with a multiple selection.

Requirement

- The HMI screen is open with at least one object.

Move objects

To move the objects, follow these steps:

1. Select the object that you want to move.
2. The following options are available for moving:
   - Moving by selecting and dragging with the mouse.
   - Moving using the arrow keys on your keyboard with pixel accuracy.
   - Coarser move using <Shift> and the arrow keys on your keyboard. The increment depends on the settings that you have configured for the grid under "Options > Settings > Visualization > Screens > Grid".

The same principle also applies to multiple selection of the objects.

Automatic scrolling

If you drag-and-drop the object to one of the corners of the screen, automatic scrolling is triggered. You can place the object in other areas of the screen.

4.3.9.3 Rotating object

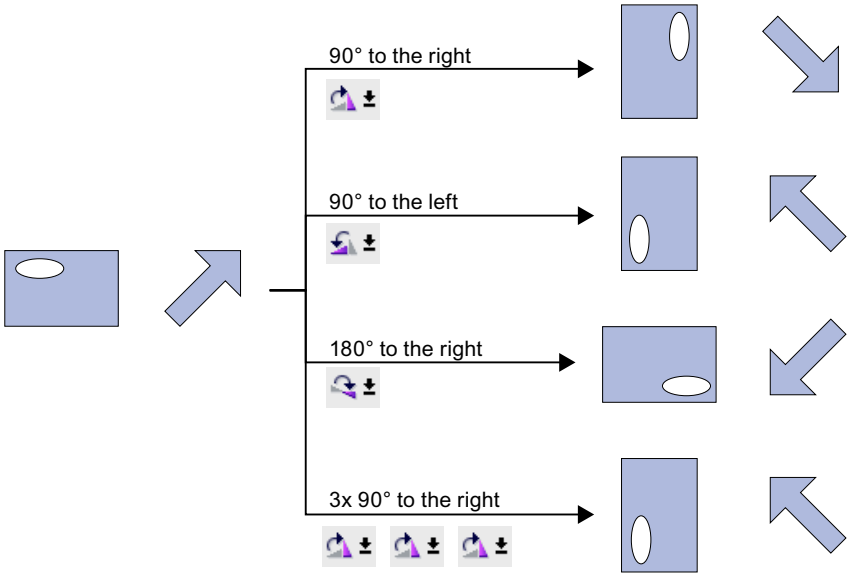
Introduction

You can rotate an object clockwise or counterclockwise around its center axis in steps of 90°.

You can also rotate multiple objects using the multiple selection function. Each object has its own reference point for rotation and is rotated around its own reference point during multiple selection.

Certain WinCC objects, e.g. controls, cannot be rotated.

The alignment of elements in an object will change in a rotated object. The following figure shows how a rectangle and an ellipse behave under the different commands for rotating an object:



Requirement

- The HMI screen is open with at least one object.

Rotating an object

To rotate an object, follow these steps:

1. Select the object that you want to rotate.
2. Click one of the following symbols on the toolbar:

	The object is rotated clockwise around its center. The angle of rotation is 90°.
	The object is rotated counter-clockwise around its center. The angle of rotation is 90°.
	The object is rotated clockwise by 180°.

The object is now displayed rotated.

Alternatively, select from the shortcut menu the desired command to rotate the objects.

4.3.9.4 Rotating an object around a pivot point

Introduction

You define the rotation of an object around a pivot point. In the Inspector window of an object, specify the coordinates of the "Pivot point X" and "Pivot point Y". Specify the angle of rotation for the object under "Properties > Rotation - angle".

The pivot point can be outside the object.

Requirement

- The HMI screen containing at least one object group is open.

Rotation

Defines the rotation of an object around the pivot point. Rotation is specified in degrees. The configured start point corresponds to a value of 0°. The position of an object deviates from its configured initial position by the rotation value. The negative and positive values are allowed.

You can also place an object outside the visible plant complex. You can view objects outside the visible area by using the "Layout > Objects out of range" task card. You change the position of an object in the Inspector window under "Properties".

Pivot point

Define the pivot point under "Properties > Rotation - Pivot point":

- Absolute to the center point: Sets the rotation to around the absolute center of the object.
- Absolute to screen: Sets the rotation to around the absolute zero point of the screen. The zero point is in the top left corner of the screen.

Rotation position

The attributes "X pivot point" and "Y pivot point" define the horizontal and vertical distance of the pivot point from the point of origin.

- Center point of the object
- Zero point of the screen

The values are specified as a device-independent pixel (DIP).

The pivot point can also be located outside the bounding box. The negative and positive values are allowed.

Example: Configuring rotation for a rectangle

To configure the rotation of a rectangle, follow these steps:

1. Open the "Basic objects" palette in the "Toolbox" task card.
2. Drag the "Rectangle" object into the screen.

3. Click "Properties > Pivot point" in the Inspector window.
4. In the "Static value" column, select "Absolute to center point".
5. Enter a value of 45 for "Rotation".

The object is rotated clockwise by 45°.

4.3.10 Designing colors

4.3.10.1 Designing the background color

Introduction

You can design the background color of an object. For certain objects, you can define a background with a color gradient.

Requirement

- The HMI screen is open with at least one object.

Designing the background color of an object

To design the background color of an object, follow these steps:

1. In the Inspector window, click "Properties > Properties > Appearance > Background - color".
2. Select a color for the background of the object, for example, yellow.

	**Efficiency tip**
Using multiple selection, you can define the background color simultaneously in multiple objects.	

Result

The object is filled with the selected color.

4.3.10.2 Defining color gradients

Introduction

You can define different color gradients for the objects. Change the category in the Inspector window, depending on which surface you fill with a color gradient.

Requirement

- The HMI screen is open with at least one object.

Configure horizontal color gradient with two colors

To configure a horizontal color gradient for an object, follow these steps:

1. Select an object, e.g. rectangle.
2. In the Inspector window, select "Horizontal gradient" under "Properties > Background - fill pattern".
3. Go to "Properties > Fill direction" and select the direction in which the color is to run, for example "Left to right".
4. Select a background color for the horizontal color gradient, e.g. orange, under "Properties > Background - color".
5. Select the other color for the gradient, e.g. yellow, under "Properties > Background - alternative color".

Result

The background of the rectangle is displayed with a color gradient of orange to yellow.

4.3.10.3 Central color management

Basic principles for central color management

Introduction

In WinCC Unified, you can change the colors that are used in a project in a centralized manner.

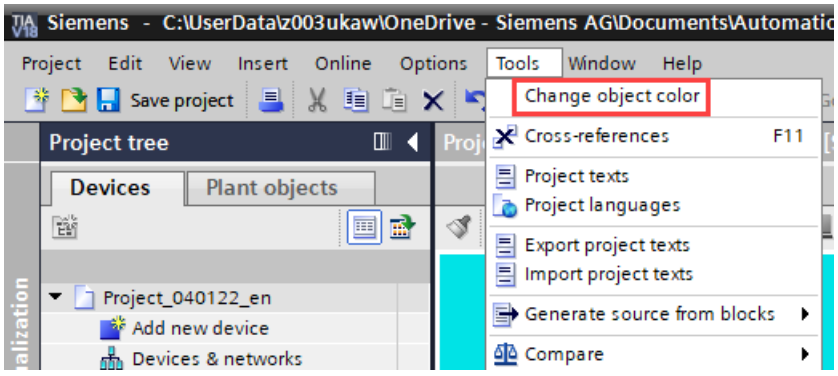
Requirement

- You have created a project.
- You have created a screen.

Opening the "Change object color" dialog box

You open the "Change object color" dialog box:

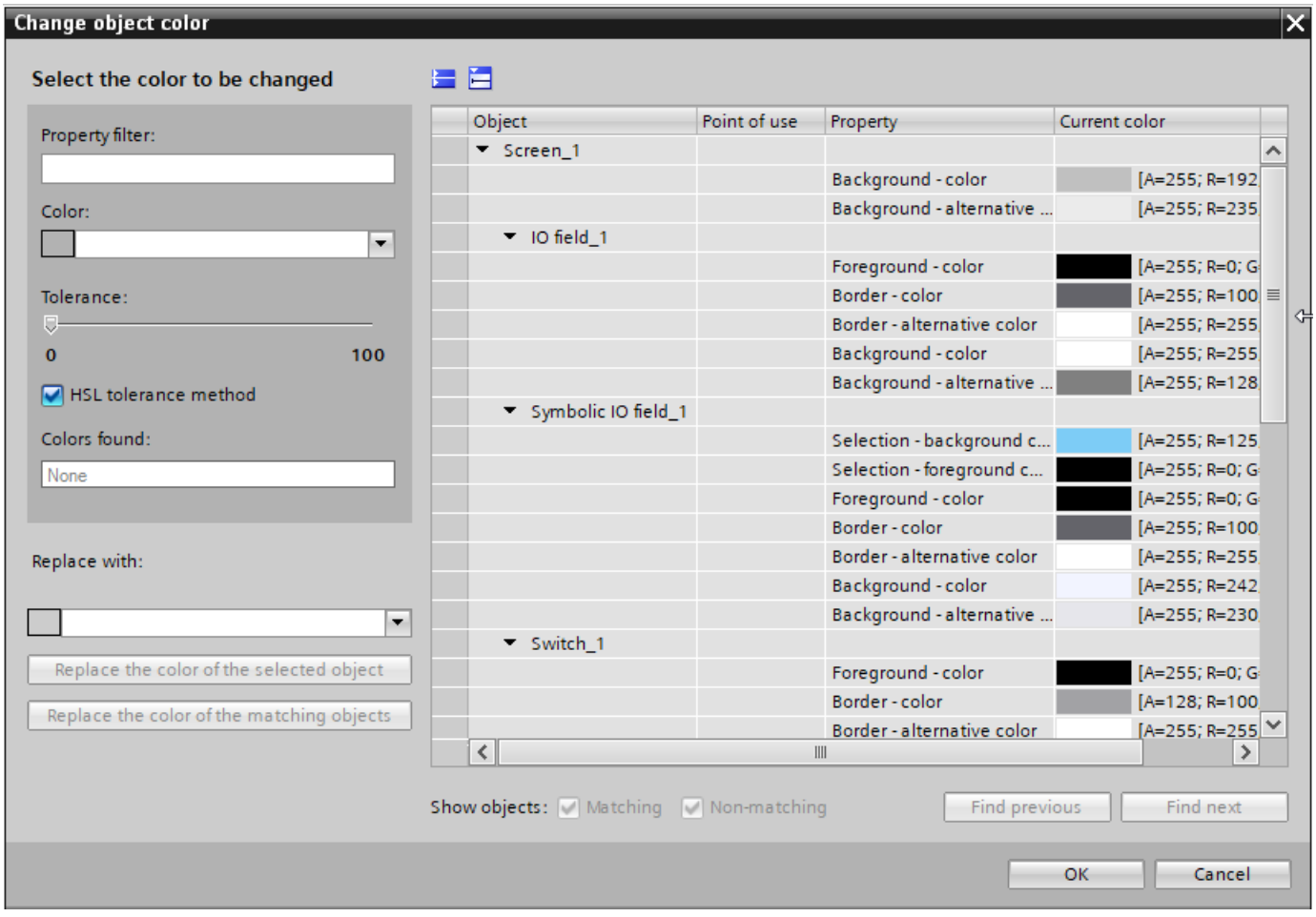
- In the shortcut menu of a device or object
- In the menu bar under "Tools > Change object color"



Using the "Change object color" dialog box

The "Change object color" dialog box contains a hierarchical overview of all color-relevant object properties.

In the display, you can navigate within the display and operating objects. You will receive an overview of all colors in use. You specify a color selection and replace it with other colors.



Unsupported objects

You have access to all colors used and configured in the project with the "Change object color" dialog.

Excluded from this are colors that are used:

- In types and instances from a library
- In scripts
- In designs
- In screens with write protection

See also

Changing the object color (Page 404)

Changing the object color

Introduction

The scope of the objects displayed in the "Change object color" dialog depends on the location at which the dialog is called:

- If you select a device and call the dialog, all color references used on the device are displayed.
- If you select an object within a screen and call the dialog, only those color references that are included in the object are displayed.

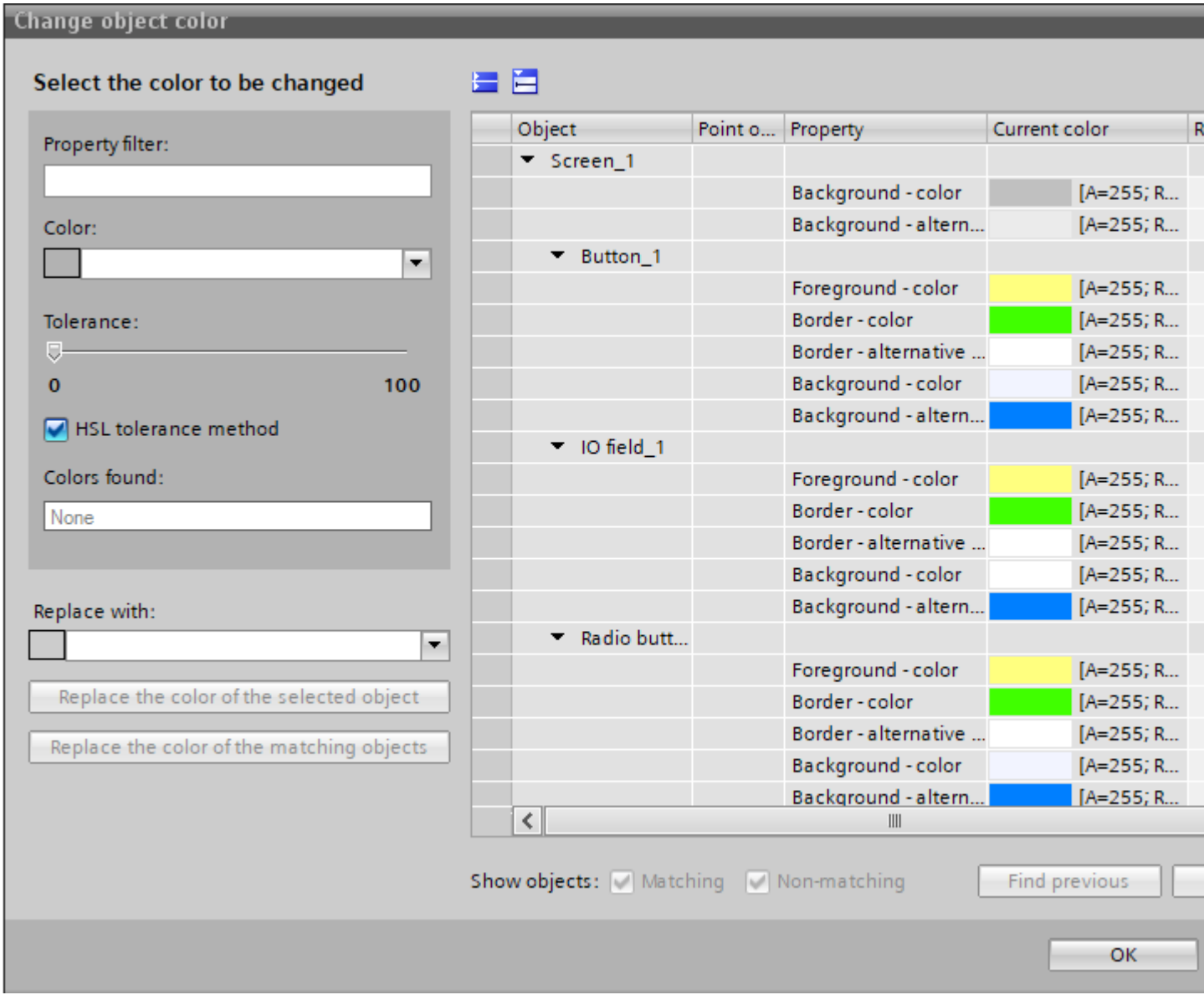
Requirement

- You have created a project.
- You have created a screen.
- At least one object has been created in the screen.

Opening the "Change object color" dialog box

To change the object color, follow these steps:

1. Select the object that contains the required color references.
2. Select "Change object color" in the shortcut menu of the object.
   The "Change object color" dialog box opens.


Change object color
Select the color to be changed
Property filter:
Color:
Tolerance:
0
100
HSL tolerance method
Colors found:
None
Replace with:
Replace the color of the selected object
Replace the color of the matching objects
Object
Point o...
Property
Current color
Screen_1
Background - color
[A=255; R...
Background - altern...
[A=255; R...
Button_1
Foreground - color
[A=255; R...
Border - color
[A=255; R...
Border - alternative ...
[A=255; R...
Background - color
[A=255; R...
Background - altern...
[A=255; R...
IO field_1
Foreground - color
[A=255; R...
Border - color
[A=255; R...
Border - alternative ...
[A=255; R...
Background - color
[A=255; R...
Background - altern...
[A=255; R...
Radio butt...
Foreground - color
[A=255; R...
Border - color
[A=255; R...
Border - alternative ...
[A=255; R...
Background - color
[A=255; R...
Background - altern...
[A=255; R...
Show objects:
Matching
Non-matching
Find previous
OK

Select the color you want to change

The following table shows you the options for choosing the color you want to change:

Select the object property whose color you want to change.	• Select a property directly in the overview. • Enter a property in the "Property filter" dialog. The selected property is now visible in the overview table.
In the "Color" text box, select the color that you want to change.	1. Click on the arrow in the "Color" text box. The project color selection opens. 2. Select a standard color. To select a user-defined color, click "More colors".
Select the color directly from the selected object.	1. Drag a color box from the "Current color" column to the "Color" text box.

Change color

To change a color, proceed as follows:

1. Select the color you want to change as described above.
2. To display the current selection, enable one or both of the options "Matching" and "Non-matching".
3. To select similar colors as well, set the tolerance. Enable the "HSL tolerance method" option. In the "Colors found" text box, you will see the number of similar colors.
4. In the "Replace with" text box, select a color.
5. Select one of the buttons:
   - "Replace the color of the selected object"
   - "Replace the color of the matching objects"

   The selected color is displayed in the "Replace with" column.
6. Click "OK".
   The dialog closes. The colors in the object are changed.

Result

You have configured new color references in the selected object.

4.3.11 Formatting text in the object

4.3.11.1 Enter text directly into the object

Introduction

You can change the label of the "Text box" and "Button" objects directly via the keyboard.

Requirement

- The HMI screen with a text box or a button is open.

Entering text directly into the object

To enter text directly in a text box or a button, follow these steps:

1. Select the object that you want to label.
2. Double-click in the object and type the text.

The text has been entered into an object.

Special features of direct text input

The following special features apply to direct input:

- Diacritics, such as ä ê ñ, can only be entered if the keyboard layout provides a key for this character. Key sequences such as <`a> for à, are not recognized.
- It is not possible to enter Unicode characters using Alt codes.
- Asian language characters cannot be entered using an Input Method Editor (IME).

If you need such characters in labeling the object, you have the following options:

- Use a keyboard layout on which this character is present as a key.
- Copy the character or full label from any source. Paste the text into the selected object.
- Edit the label in the Inspector window under "Properties > Properties > General > Text".

Direct text entry also supports multiline text. Enter the line break with the keyboard shortcut <Shift + Enter>.

See also

Text box (Page 269)

Button (Page 289)

4.3.11.2 Entering multiline text

Introduction

For objects with the "Text" property, you can enter the text on multiple lines.

Requirement

- The HMI screen with at least one object with the "Text" property is open.

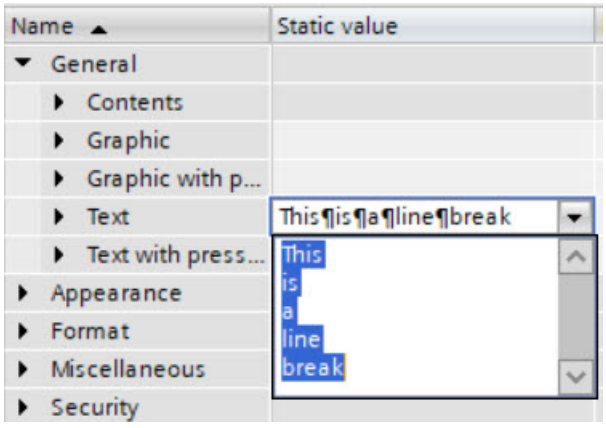
Enable line break

Enter the line break using the <Shift + Enter> key combination.

Multiple line text for objects

Enter the multiple line text for the following objects in the Inspector window under "Properties > Properties > General > Text":

- Text box
- Button
- Switch



Enter the multiple line text for the following objects under "Properties > Properties > General > Title > Text".

The text may only contain 2 lines at a time.

- Bar
- Slider

For the following objects, enter the multiple line text under "General > Selection items > [x] Selection item> Text".

The text may contain more than 2 lines if the property "Format > Item height" is adjusted accordingly.

- Check box
- Radio button
- List box

You can use multiple line texts with a maximum of two lines in the text list for symbolic IO fields.

Alternatively, enter the multiple line text

Alternatively, enter the multiple line text:

- In the Inspector window, under "Properties > Text".
  Enter the line break using the <Shift + Enter> key combination. A line break is displayed as a blank.

- By entering text directly into the object for:
  - Text box
  - Button

4.3.11.3 Show default entry of text and graphic list in the object

Introduction

For objects with the dynamization type "Resource list", you can have a text or a graphic from the resource list displayed as the default entry in the object.

Requirement

- The HMI screen with an object, e.g. a button, is open.
- At least one object property supports the "Resource list" dynamization type, for example, "text" or "graphic".

Displaying a text as the default entry in the object

To display a text as the default entry in the object, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Text".
2. Select "Resource list" in the "Dynamization" column.
3. In the "Resource list" dialog, select a text list under "Settings > Resource list".

4. Click on the button in the "Resource list" line .
   The selected text list opens.
5. Select an entry in the "Text list entries" table as the default entry. The text from the default entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Text" column is displayed in the object.

Displaying a graphic as the default entry in the object

To display a graphic as the default entry in the object, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Graphic".
2. Select "Resource list" in the "Dynamization" column.
3. In the "Resource list" dialog, select a graphic list under "Settings > Resource list".
4. Click on the button in the "Resource list" line .
   The selected graphic list opens.
5. Select an entry in the "Graphic list entries" table as the default entry. The graphic from the standard entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Graphic" column is displayed in the object.

See also

Configuring object with a text list (Page 453)

4.3.11.4 Displaying tag value in the object dynamically

Introduction

In some objects, dynamic information can be displayed in the properties that contain a text.

If you insert the reference to a tag in the "Text" or "Tooltip" property, for example, the current value of the inserted tag is displayed in the object in Runtime.

Supported objects:

- Text box
- Button
- Symbolic IO field
- Switch

Note

In the message texts, e.g. for the alarm control, the dynamic information may already be inserted.

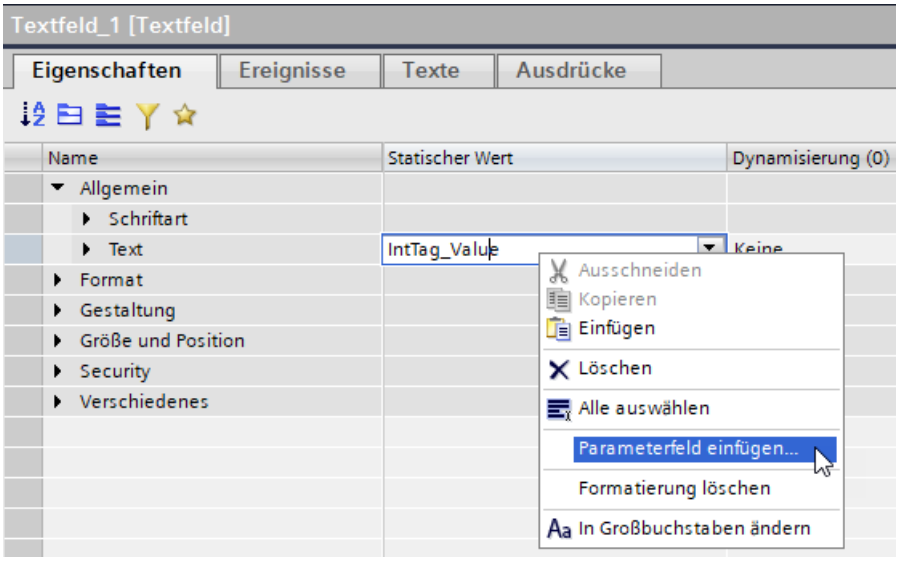
Requirement

- The HMI screen with an object, e.g. a text box or a button, is open.

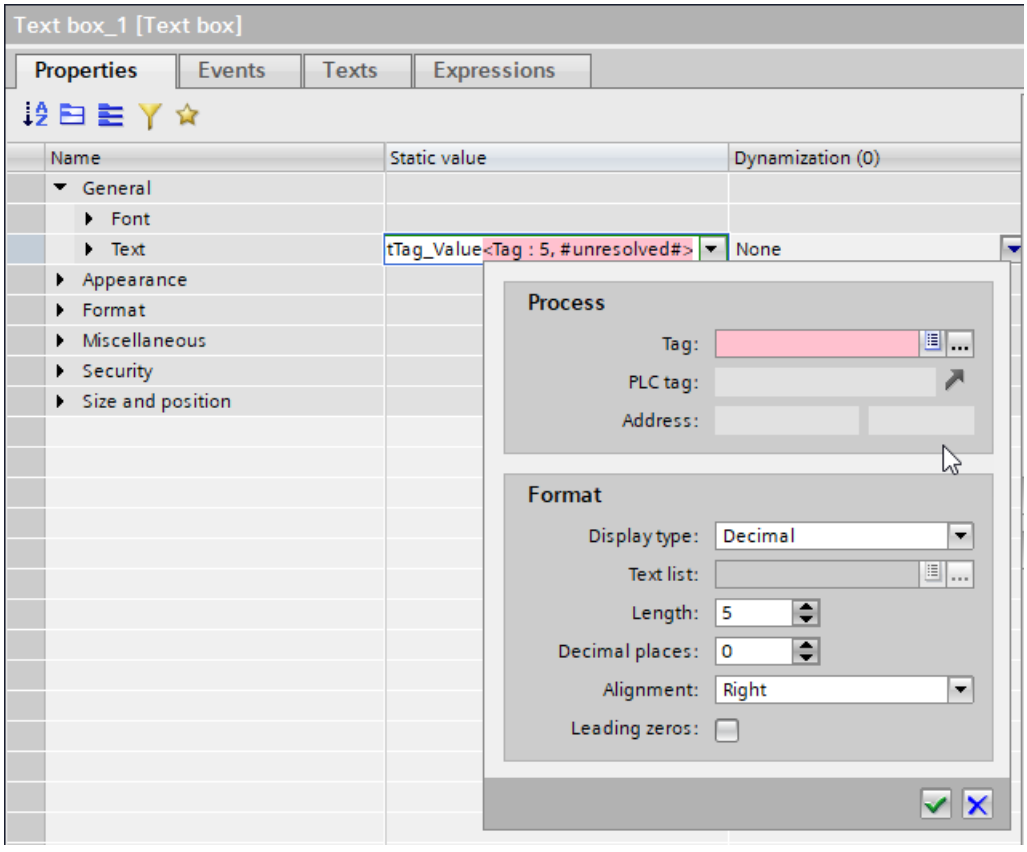
Configuring a tag value in the object

To configure a tag value in an object, e.g. in a text box, follow these steps:

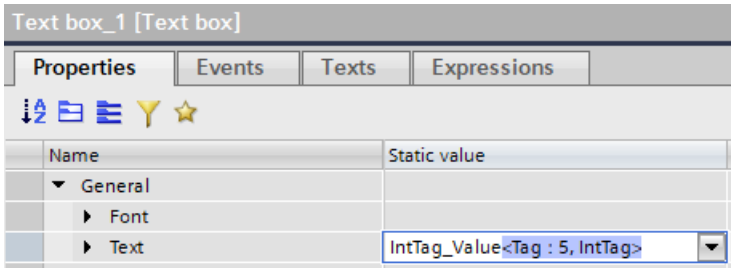
1. Configure a tag, e.g. "IntTag".
2. In the Inspector window, select "Properties > Properties > General > Text".
3. Enter a text in the "Static value" column in the input field, e.g. "IntTag_Value".
4. Right-click in the input field. The shortcut menu opens.



5. Select "Insert parameter field" in the shortcut menu. A dialog opens.



6. Select the "IntTag" tag under "Tag" in the dialog.
7. Configure the format of the display:
   - Display type
   - Length
   - Decimal places
   - Alignment:
   - Leading zeros
8. Confirm the selection. The field with the tag name becomes visible in the input field.



9. Compile and load the device.

Note

If you have inserted the parameter field with a tag in an object, you cannot edit the text in the object directly. After deleting the parameter field, you can edit the text in the object directly again.

More information on direct text input is available in Enter text directly into the object (Page 406).

Result

The object displays the current tag value in Runtime.

4.3.11.5 Dynamically displaying a text list in the object

Introduction

In some objects, dynamic information can be displayed in the properties that contain a text.

If you insert a reference to a text list in the "Text" or "Tooltip" property, for example, the current value of the inserted text list is displayed in the object in Runtime.

Supported objects:

- Text box
- Button
- Symbolic IO field
- Switch

Note

In the message texts, e.g. for the alarm control, the dynamic information may already be inserted.

Requirement

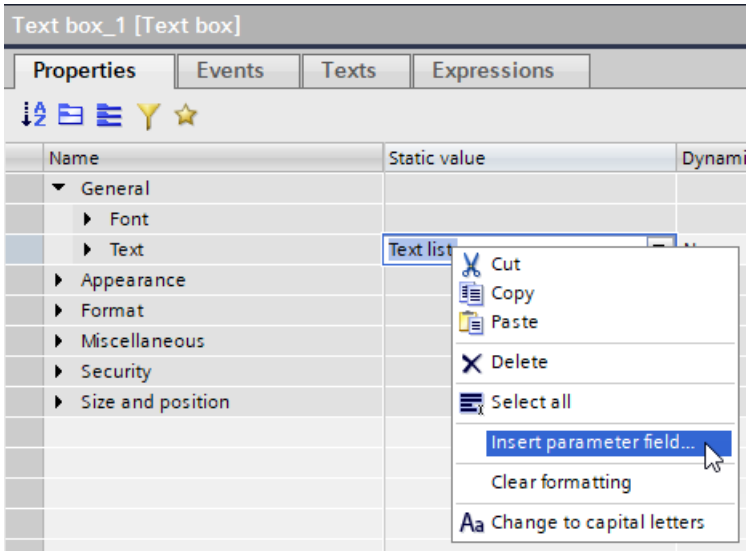
- The HMI screen with an object, e.g. a text box or a button, is open.

Configuring a text list in the object

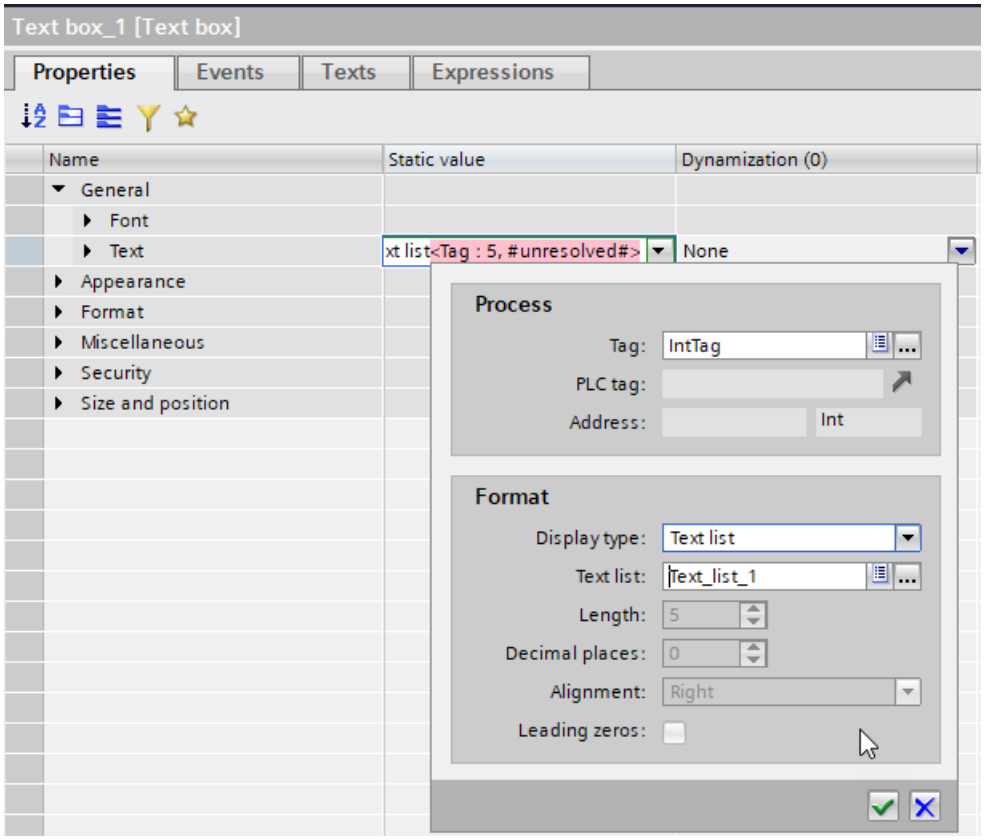
To display a text list in an object, e.g. in a text box, follow these steps:

1. Configure a tag, e.g. "IntTag".
2. Configure a text list, e.g. "Textliste_1".
3. In the Inspector window, select "Properties > Properties > General > Text".
4. Enter a text in the "Static value" column in the input field, e.g. "Text list".

5. Right-click in the input field. The shortcut menu opens.

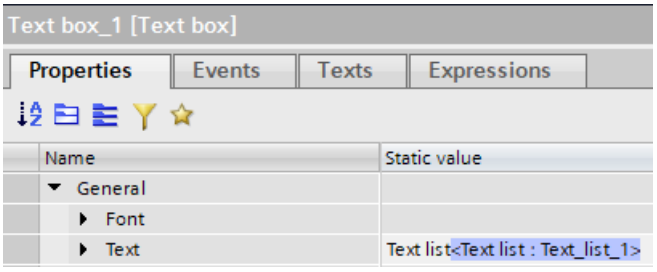


6. Select "Insert parameter field" in the shortcut menu. A dialog opens.



7. Select the "IntTag" tag under "Tag" in the dialog.
8. Under "Display type", select "Text list".
9. Under "Text list", select the text list "Textlist_1".

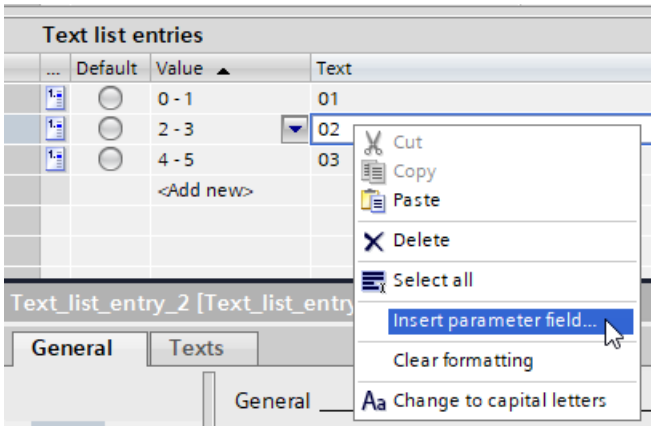
10. Confirm the selection. The field with the tag name becomes visible in the input field.



The default entry of the text list is displayed in the text box.

11. Compile and load the device.

Alternatively, you can configure the text list entry directly in the text list.



Note

Circular references are not supported.

Result

The object displays the current text list entry in Runtime.

4.3.12 Linking objects

4.3.12.1 Linking an object to a text list

Introduction

You can link the objects to a text list.

Requirement

- The HMI screen with an object is open.

Linking objects to a text list

You can link a text list to the following objects:

- Text box
- Button
- Switch
- Symbolic IO field

To link a text list to an object, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Text".
2. Select the "Resource list" option in the "Dynamization" column. The "Resource list" page opens.
3. Select the text list from which the text is displayed.

The text list has been linked to an object.

	Tips for an efficient procedure
•	Use drag-and-drop to move a text list from the detail view of the text and graphic lists directly onto an object in a screen. The object is linked to the text list.

4.3.12.2 Linking an object to a graphic list

Introduction

You can link the objects to a graphic list.

Requirement

- The HMI screen with an object is open.

Linking objects to a graphic list

You can link a graphic list to the following objects:

- Graphic view
- Button
- Switch
- Symbolic IO field

To link a graphic list to an object, follow these steps:

1. In the Inspector window, select "Properties > Properties > General > Graphic".
2. Select the "Resource list" option in the "Dynamization" column. The "Resource list" page opens.
3. Select the graphic list from which the graphic is displayed.

The graphic list has been linked to an object.

	**Tips for an efficient procedure**
• Use drag-and-drop to move a graphic list from the detail view of the text and graphic lists directly onto an object in a screen. The object is linked to the graphic list.	

4.3.12.3 Linking an object to tags

Introduction

You can link the objects to a tag.

Requirement

- The HMI screen with an object is open.

Link objects to a tag

You can link a tag to the following objects:

- Text box
- IO field
- Symbolic IO field
- List box
- Bar
- Slider
- Gauge
- Clock
- Check box
- Radio button
- Trend control
- Process control

To link a tag with an object, proceed as follows:

1. In the Inspector window, select "Properties > Properties > General > Process value".
2. Select "Tag" in the "Dynamization" column. The "Tag" page will open.
3. Select an existing tag under "Tag > Process > Tag".
   Alternatively, create a new tag using the "Add" button.

The tag has been linked to an object.

	Tips for an efficient procedure
•	Drag a tag from the detail view of the tag table directly to an object in a screen. The object is linked to the tag.

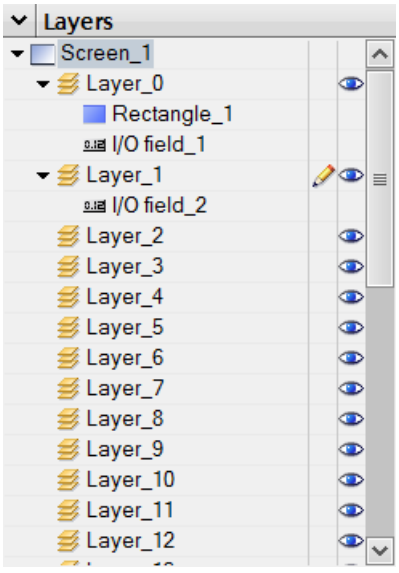
4.3.13 Using layers

4.3.13.1 Basic information on using layers

Layers

Use layers in order to achieve differentiated editing of the objects in a screen. Using layers, multiple objects can be combined and edited together, for example. Layers are also used to improve clarity during configuring, because multiple objects can be hidden and displayed again when required.

A screen has 32 layers. The name of the individual layers is determined by the user interface language and changes when the user interface language is changed. If you assign objects to the layers, you thereby define the screen depth. Objects of layer 0 are located in the screen background, while objects of layer 31 are located in the foreground.



The objects of a single layer are also arranged hierarchically. If you create a new object, it is arranged in the foreground. You can shift objects forwards and backwards within a layer.

Principle of the layer technique

Always one layer of the 32 layers is active. New objects you add to the screen are always assigned to the active layer. The active layer is indicated in the "Layout > Layers" task card.

When you open a screen, all 32 layers of the screen are displayed. You can hide all layers except for the active layer in the "Layout > Layers" task card. You then explicitly edit objects of the active layer.

In the "Layout > Layers" task card, you can also manage layers and objects with drag-and-drop and the shortcut menu.

Application examples

Use layers, for example, in the following cases:

- To hide the labeling of objects when editing,
- To hide individual objects, while you configure other objects

4.3.13.2 Renaming a layer

Introduction

When you create a screen, the 32 layers are numbered consecutively by default. To improve clarity, you can rename the layers to suit your requirements.

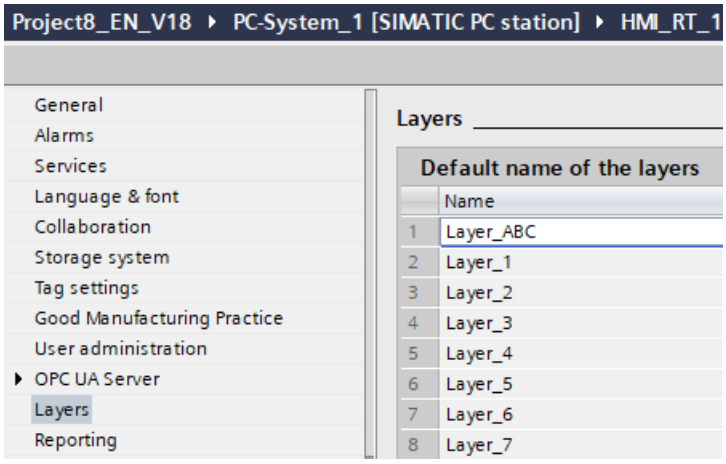
Requirement

- A screen with an object is open.

Renaming a layer

To rename the name of a layer, follow these steps:

1. Click on Runtime settings in the project tree.
2. In the "Name" column, click in the row of the layer.
3. Enter a name.



Result

The selected layer has been renamed.

4.3.13.3 Moving objects between layers

Introduction

By default, newly inserted objects are in the foreground of the active layer. You can assign an object to a different layer and change the order of objects within a layer at a later time.

Requirement

- A screen with an object is open.

Moving objects between layers

To move the objects, follow these steps:

1. Select the object in the "Layout > Layers" task card.
2. Drag-and-drop the object to the required layer.

Changing the order of objects within a layer

To change the sequence of objects, follow these steps:

1. Select the object in the screen.
2. Select the desired command under "Arrange" in the shortcut menu. Depending on the current position of the object, you can move it completely into the foreground, to the front, to the back or completely into the background.

Result

The object is arranged according to the selection. In the "Layout > Layers" task card, the order of the objects is displayed as follows: Objects of layer 0 are located in the screen background, while objects of layer 31 are located in the foreground. Within a layer, the objects displayed at the top of the list are in the background of the layer.

4.3.13.4 Specifying the active layer

Introduction

The screen objects are always assigned to one of the 32 layers. There is always an active layer in the screen. New objects you add to the screen are always assigned to the active layer.

The active layer is indicated by a  icon in the "Layout > Layers" task card.

You can activate a different layer during configuration, if necessary.

Requirement

- You have opened a screen which contains at least one object.

Procedure

To set a layer as active, follow these steps:

1. Select "Layout > Layers" in the "Layout" task card.
2. Select the "Set to active" command from the shortcut menu of a layer.

Result

The selected layer becomes the active layer.

4.3.13.5 Hiding and showing layers

Introduction

You can show or hide the layers of a screen as required.

Requirement

- The screen is open.

Hiding or showing layers

To hide or show layers, follow these steps:

1. Select the layer that you want to hide or show in the "Layout > Layers" task card.
2. Click one of the icons next to the corresponding layer:

- A shown layer is hidden.
- A hidden layer is shown.

You cannot hide the active layer.

Alternatively, select the "Hide layer" or "Show layer" command from the shortcut menu of a layer.

Result

Only the displayed layers are shown in the Engineering System.

Setting the visibility of the levels in the Engineering System has no influence on the visibility of the levels in Runtime.

4.3.13.6 Toggle the visibility of layers in runtime in the ES

Introduction

You can toggle the visibility of layers in runtime in the Engineering System.

Requirement

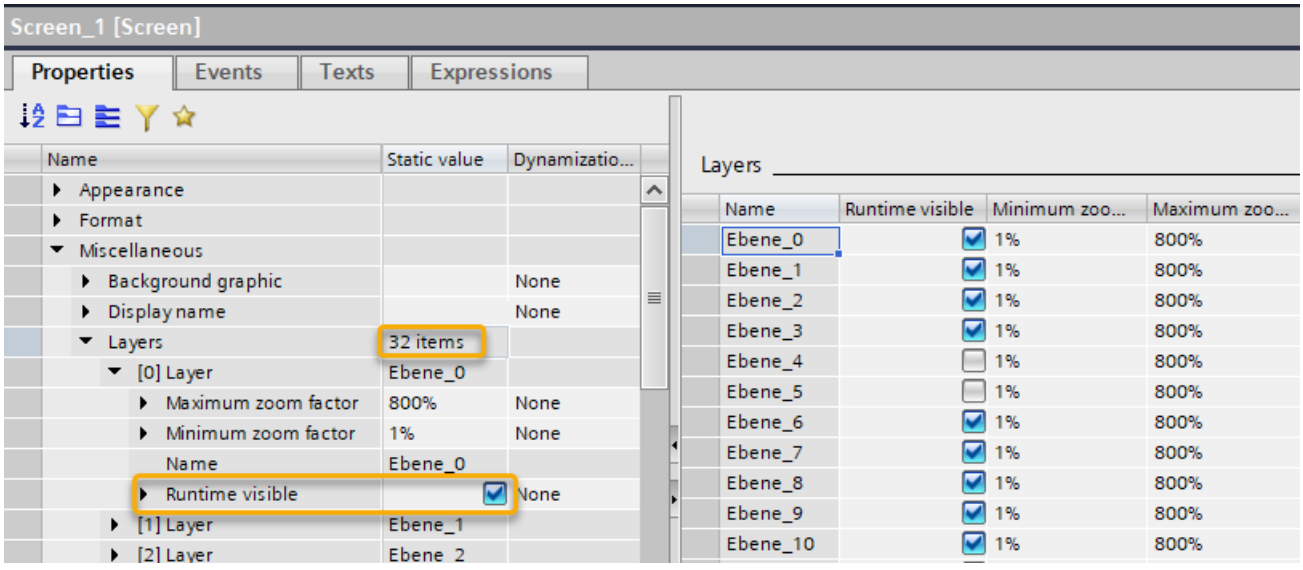
- The screen is open.

Toggling the visibility of layers in the Engineering System

To switch the visibility of layers in runtime in the Engineering System, follow these steps:

1. In the Inspector window, select "Properties > Properties, Miscellaneous > Layers".
2. Select the layer whose runtime visibility you want to toggle.

3. Under "Miscellaneous > Levels > [x] Level", select the "Runtime visible" option in the "Static value" column.



4. If you now click on the field "Layers" under "Miscellaneous", the overview of all layers becomes visible in the right part of the Inspector window.
   You can toggle the visibility directly in the overview.

Result

You can now toggle the visibility of the layers in Runtime.

Only the displayed layers are shown in the Engineering System.

Setting the visibility of the levels in runtime has no influence on the visibility of the levels in the Engineering System.

4.3.13.7 Toggling the visibility of layers in runtime using the JScript function

Introduction

You can toggle the visibility of layers in runtime using the JScript function.

Requirement

- The screen is open.

Toggling the visibility of layers via the JScript function

To toggle the visibility of the layers in runtime using a JScript function, follow these steps:

1. In the Engineering System, place the objects of a screen in the "Layout" task card on different layers.
2. Program a JScript function to change the visibility of, for example, "Layer_1" in the current screen.
   ```
   Screen.Layers ("Layer_0").Visible = false;
   ```
3. Configure this function, e.g. on an event of a button.

If the event occurs in runtime, all objects placed on this layer become invisible.

To make a layer visible in the current screen, use:
```
Screen.Layers("Layer_0").Visible = true;
```

Result

You can change the visibility of the layers in Runtime.

4.3.14 Using groups

4.3.14.1 Basics of groups

Introduction

You can put two or more objects together to form a group with the "Group" function.

You can manage a group of objects in the Engineering System and in Runtime just like you manage an individual object, e.g. you can change the color or visibility of all grouped objects in one step.

Grouping is available on a Unified PC and Unified Control Panel as of version V18. The grouping of objects is not supported in the VoT application.

Editing objects together

You can edit multiple objects together by means of:

- Multiple selection (Page 361): With multiple selection, the bounding boxes of all objects are displayed.
- Grouping objects (Page 426): With a group, one bounding box is displayed for the whole group.

Groupable objects

You can add the following objects to a group:

- Basic objects
- Elements
- Faceplate containers
- Graphics
- Dynamic widgets

Nesting of groups – group in group – is not supported.

Non-groupable objects

You cannot add the following objects to a group:

- Controls (except faceplate containers)
- Custom controls

Layers

All objects of a group are located in the same layer. The groups are arranged hierarchically in a layer.

You can also add objects to a group or remove them from a group in the "Layout > Layers" task card using drag-and-drop.

Properties of a group

- The group has its own coordinate system. The coordinates of all objects contained in the group are referenced to the upper left corner of the group.
- Each group has its own properties. You can define the properties of a group for the runtime:
  - Via a dynamic property.
  - Via a script.
- If you set a property for a group, all objects of the group inherit this property.
- If you configure the same property differently on an object in the group, the property value on the object is valid in runtime, not the value in the group.
- When you resize an object, the group is also resized.
- The grouping of objects is also possible within faceplate types.
- Groups have no events for which you can configure event scripts or function lists. The events of the objects in the group work in the same way as for objects that are not grouped.
- Nesting of groups – group in group – is not supported.

4.3.14.2 Grouping objects

Introduction

The "Group" menu command combines multiple objects to form a group.

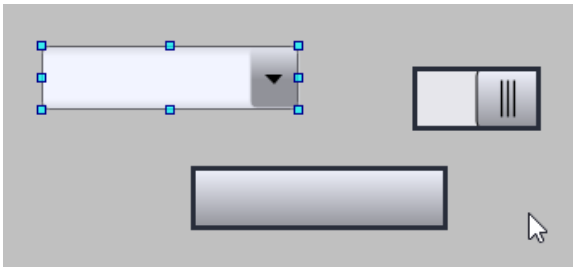
Requirement

- A screen containing at least two objects is open.

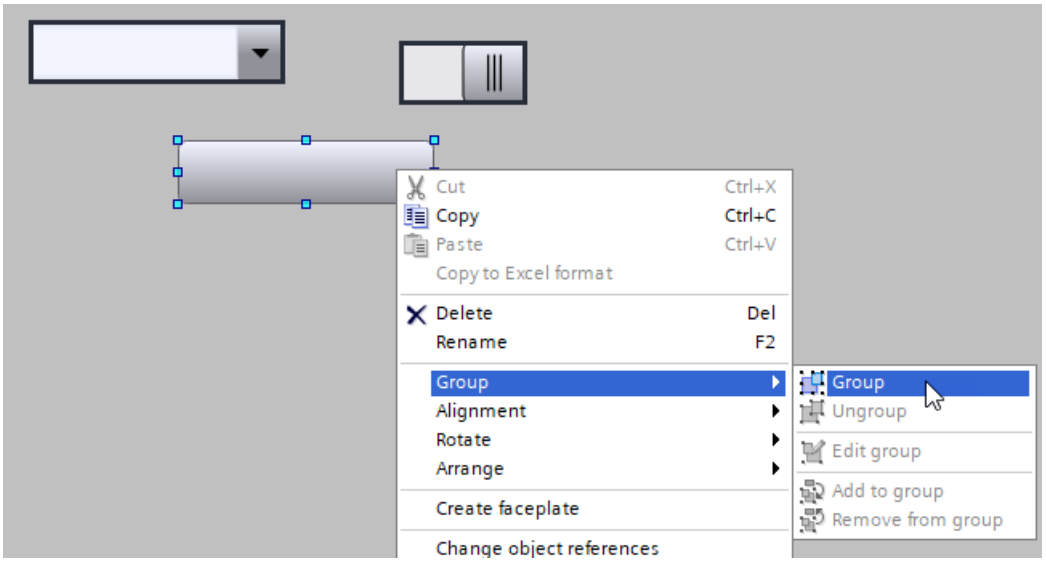
Grouping objects

Follow these steps to group objects:

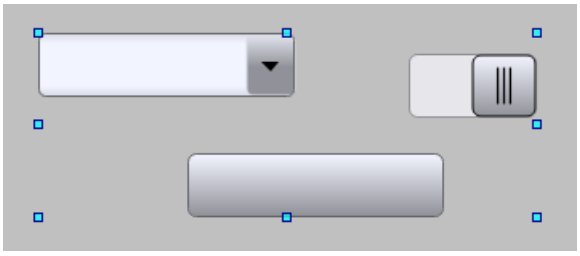
1. Select all the objects you want to group using multiple selection.



2. Select the "Group > Group" command from the shortcut menu.



3. The group objects are displayed with a bounding box.



Result

The selected objects are combined in a group. The multiple selection bounding box becomes the group bounding box. The resizing handles are shown only for the group.

The group is in the active layer.

The group is given a unique default name, e.g. "Group_1".

The coordinates of the upper left corner of the group are defined in such a way that the position of the objects contained in the group remains unchanged in the screen. The height and width of the group is defined to accommodate the full extent of the objects contained in the group.

4.3.14.3 Managing groups

Introduction

You can select, copy, paste and ungroup a group.

Requirement

- A screen containing at least one group of objects is open.

Selecting a group

When you click on an object in the group, the whole group is selected.

If the whole group is visible in the screen editor, you can also select the group by drawing a lasso around it.

Copying and pasting a group

To copy a group, follow these steps:

- Right-click the group in the "Layers" task card. Select "Copy" and "Paste" in the shortcut menu.
- Select the group. Copy the group with <Ctrl + C> and paste it with <Ctrl + V>.

You can paste the copied group:

- In the same screen
- In a different screen
- On another Unified device

Note

References in function lists or scripts to other objects in the group are kept as the original reference.

Example:

Group_1 contains Button_1 and Circle_1.

Button_1 has a script that references Circle_1.

After copying and pasting, the copied Group_2 contains Button_2 and Circle_2.

The script on Button_2 continues to reference Circle_1.

Ungrouping a group

To ungroup a group, follow these steps:

1. Select the group.
2. Right-click to open the shortcut menu.
3. In the shortcut menu, select the "Group > Ungroup" command.

The group is ungrouped. The objects from the group remain in the screen.

4.3.14.4 Changing the size of the group

Introduction

You can change the size of an object group in the Engineering System.

When you select the group, it is framed by a bounding box with blue handles. You have the following options for changing the size and position of a group:

- Drag with the mouse
- Configure group properties

After the size of the group has been changed, the objects contained in the group are scaled proportionally to their original size and the target size.

Requirement

- The HMI screen containing at least one object group is open.

Changing the size of the group with the mouse

To change the size of the group with the mouse, follow these steps:

1. Select the group whose size you want to change.
   The bounding box is displayed.
2. Drag a handle of the box to a new position.

The size of the group is changed.

	**Tips for working effectively**
If you press the <Shift> key while dragging, the group is resized according to the aspect ratio.	

Note

- Circular objects, such as circles, circular arcs, circle segments, gauges, and clocks, are only resized if both dimensions are changed at the same time by dragging one of the corner handles. If the size of the group is not adapted proportionally, the circular objects cannot be displayed correctly.
- If a group was first significantly reduced in size and then enlarged again, the objects in it may shift or reshape due to precision losses.

Configuring the size of the group through properties

To change the size of the group through properties, follow these steps:

1. Select the group whose size you want to change.
2. Select "Properties" > "Properties" > "Size and position".
3. In the "Static value" column, enter the "Size - width" and "Size - height" coordinates.

The object size is changed.

4.3.14.5 Moving a group

Introduction

You can move an object group in various ways.

Requirement

- The HMI screen containing at least one object group is open.

Moving an object group

To move the object group, follow these steps:

1. Select the object group you want to move.
2. The following options are available for moving:
   - Moving by selecting and dragging with the mouse.
   - Moving with pixel accuracy using the arrow keys on your keyboard.
   - Moving in larger pixel increments using <Shift> and the arrow keys on your keyboard. The increment depends on the settings you configured for the grid under "Options > Settings > Visualization > Screens > Grid".

4.3.14.6 Moving groups between layers

Introduction

You can move groups between layers in the "Layout > Layers" task card and change the order of groups within a layer.

The objects in the groups are moved to the same layer as the parent group and keep their order.

Requirement

- The HMI screen containing at least one object group is open.

Moving groups between the layers

You have the following options for moving one or more groups between the layers:

- Moving one group to another layer using drag-and-drop
- Selecting the "Arrange" command in the shortcut menu of the group, and then selecting:
  - "Bring to front"
  - "Bring forward"
  - "Send backward"
  - "Send to back"
- Moving multiple groups using multiple selection and drag-and-drop.

Note

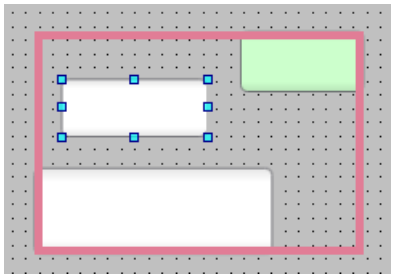
Moving of groups is only possible if no objects contained in the group have been selected.

4.3.14.7 Groups in editing mode

Introduction

You can configure the individual objects in the group in editing mode.

When you select an object within the group, the group goes into editing mode. When the group is in editing mode, it is framed by a red rectangle.



Requirement

- The HMI screen containing at least two objects is open.
- The objects have been put together to form a group.

Activating editing mode

To activate editing mode for a group, choose one of the following options:

- Click an object in the group with the left mouse button. Select "Group > Edit group" in the shortcut menu.
- Double-click an object in the group.
- Select an object within a group by clicking on the object in the "Layout > Layers" task card.
- An object in the group can be selected through an error message, cross-reference or search result.

Options in editing mode

- When the group is in editing mode, you can select an object contained in the group by clicking it. The group remains in editing mode.
- If you select an object outside the group, the editing mode is deactivated.
- You can also select the following using multiple selection:
  - Multiple objects in a group.
  - Multiple groups in the screen.
- Direct text input: If you double-click an object in which a direct text input is possible, you can enter the text. If the group is not in editing mode, editing mode is activated.
- A rotated group is displayed as not rotated in editing mode. After editing mode is deactivated, the group is displayed rotated again. The configured value of the "Rotation - angle" property remains unchanged.

Configuring objects in editing mode

You can configure the objects in the group in editing mode in the following ways:

- Changing properties of the selected objects.
- Aligning or distributing objects or setting them to the same size.

- Adding or removing points of point-based objects, e.g. for polyline and polygon.
- Copying objects. When pasted, the objects are placed outside the group.
- Linking objects, e.g. with a tag, using drag-and-drop.

The group is updated after size or position changes of the objects it contains.

See also

Aligning objects in the group (Page 436)

4.3.14.8 Adding an object to the group

Introduction

You can add one or more objects to a group.

Requirement

- The HMI screen containing at least one object group is open.
- At least one object has been configured outside the group.

Adding an object to a group

To add an object to a group, follow these steps:

1. Select a group and one or more objects using multiple selection.
2. Right-click to open the shortcut menu.
3. In the shortcut menu, select the "Group > Add to group" command.

Note

The command is hidden if you have selected unsupported objects, e.g. a control or another group.

Result

The object(s) are added to the group. The size of the bounding box of the group is adjusted to accommodate the added objects.

The added objects are displayed in the "Layout > Layers" task card at the bottom position of the group.

Adding an object to a group using drag-and-drop

You can also add objects to a group or remove them from a group in the "Layout > Layers" task card using drag-and-drop.

4.3.14.9 Managing objects in groups

Introduction

You can move or delete one or more objects in the group or remove one or more objects from the group.

Requirement

- A screen containing at least one group of objects is open.

Moving an object in the group

You can move the objects within a group in the "Layout > Layers" task card using drag-and-drop.

If you move an object directly to the group using drag-and-drop, the object is positioned last within the group.

Note

You can also add objects to a group or remove them from a group in the "Layout > Layers" task card using drag-and-drop.

Deleting an object in the group

It is not possible to delete an object that is contained in a group.

To delete an object, follow these steps:

1. Move the object directly to a layer using drag-and-drop.
2. Delete the object.

Removing an object from the group

To remove an object from a group, select one of the following options:

- Drag-and-drop the object from the group to the "Layers" task card.
  If you move an object directly to a layer using drag-and-drop, the object is positioned last within the level.
-

You cannot remove the last object from the group because an empty group is not permitted.

4.3.14.10 Rotating a group and objects in the group

Introduction

You can rotate the group and the objects in the group.

You can define the rotation of an object around a pivot point. You can specify the "Pivot point X" and "Pivot point Y" coordinates of the pivot point in the Inspector window of the group. You can specify the rotation angle for the rotation of the group under "Properties > Rotation - angle".

The pivot point can also be located outside the group.

Requirement

- The HMI screen containing at least one object group is open.

Rotating an object group

The rotation defines the rotation of the group around the pivot point. The value of the rotation is specified in degrees. The configured initial position corresponds to a value of 0°. The position of the group differs from its configured initial position by the rotation value. Negative and positive values are permissible.

Pivot point

Define the pivot point under "Properties > Rotation - pivot point":

- Absolute to center: Specifies that the rotation is an absolute rotation around the center point of the group.
- Absolute to screen: Specifies that the rotation is an absolute rotation around the zero point of the screen. In this case, the zero point is located at the top left corner of the screen.

Pivot point positioning

The attributes "Pivot point X" and "Pivot point Y" define the horizontal and vertical distance of the pivot point from the origin point.

- Center point of the group
- Zero point of the screen

The values are specified in device-independent pixels (DIP).

The pivot point can also be located outside the bounding box. Negative and positive values are permissible.

Result

The group is rotated. All objects in the group are rotated according the rotation of the group.

Rotated objects in the group

You have the following options:

- You can create a group of the rotated objects.
- You can rotate the objects in the group.

Pivot points X and Y are viewed relative to the group, if pivot point mode "Absolute to screen" is selected.

Limitations

- The rotation of objects contained in the group is not taken into account when calculating the group boundaries or when changing the size of a group.
- When you edit the objects in a rotated group, the group is displayed non-rotated while it is in editing mode. After editing mode is exited, the group is displayed rotated. The configured value of the "Rotation - angle" property remains unchanged.
- The rotation of a group is not taken into account when adding and removing objects to and from the group.
- Based on these limitations, the following is recommended:
  - Configuring a group and its objects without rotation.
  - Use a dynamic rotation in runtime when needed.

4.3.14.11 Aligning objects in the group

Introduction

You can align the selected objects in relation to a reference object in a group.

Requirement

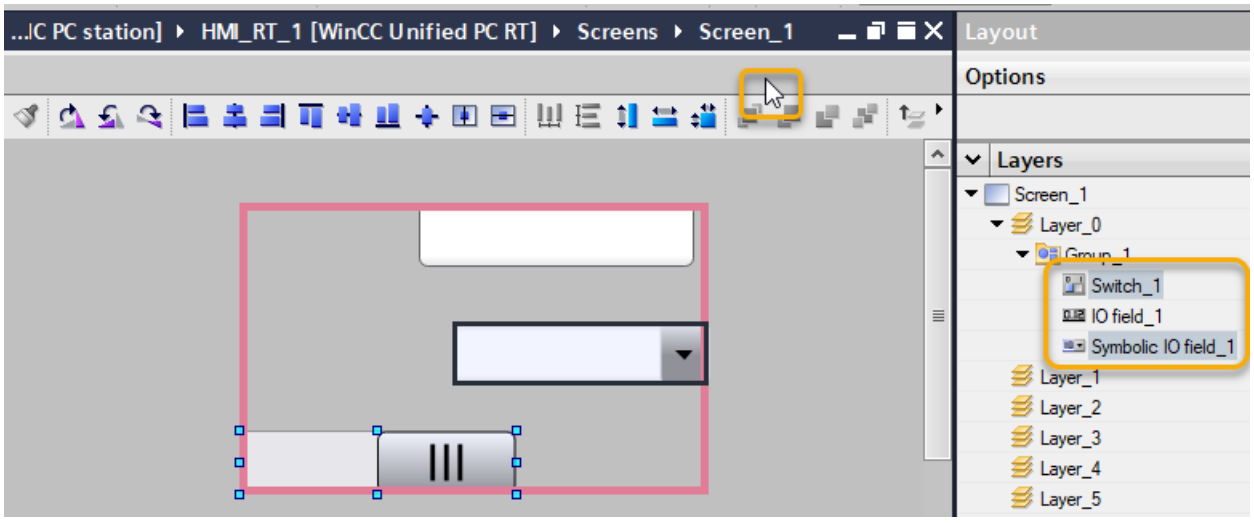
- The HMI screen containing at least two objects is open.
- The objects have been put together to form a group.

Aligning selected objects in a group

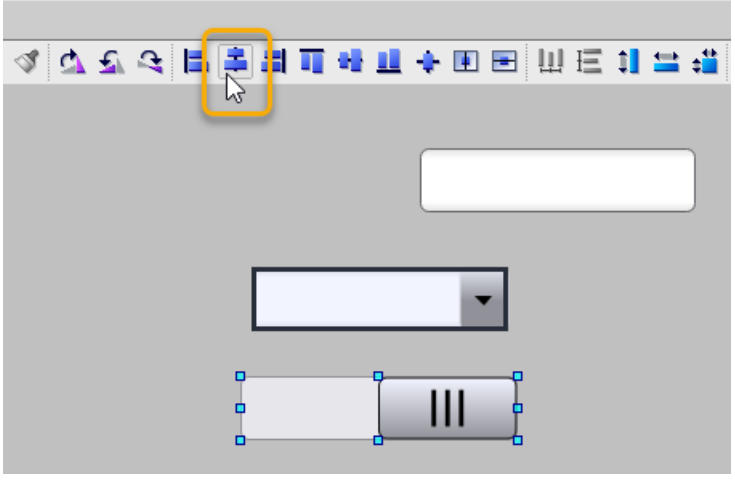
The reference object is the object that you selected first.

To align selected objects in a group in relation to a reference object, follow these steps:

1. Select the desired objects in the "Layout > Layers" task card.
2. Click on the bar above the toolbar.
   The icons in the toolbar are displayed.



3. Select the desired command for alignment in the toolbar.



The selected objects in the group are aligned in relation to the reference object.

See also

Aligning objects (Page 395)

Groups in editing mode (Page 431)

4.3.14.12 Properties of the group

Introduction

Properties of the group are properties that you edit at the group level.

You can configure the group properties in the Engineering System and interpret them in runtime.

Properties of the group

The following properties affect the entire group.

- Name
- Coordinates "Position - left" and "Position - top". The coordinates of all objects contained in the group are referenced to the upper left corner of the group.
- "Size - height" and "Size - width" form the outer boundary of the objects contained in the group.
  These values are automatically updated when:
  - Objects are added to the group.
  - Objects are removed from the group.
  - The position of the objects contained in the group is changed.
- Rotation-related properties:
  - "Rotation - pivot point"
  - "Rotation - angle"
  - "Rotation - pivot point X"
  - "Rotation - pivot point Y"

  If the group is rotated, all objects contained in the group are displayed rotated.
- Visibility: The visibility is only toggled in runtime. All objects are always visible during engineering.
  The visibility functions in a cascaded manner:
  - If the visibility of the group has been set to "false", all objects are invisible.
  - If the visibility of the group has been set to "true", the objects for which visibility is set to "true" are visible.
- When "Operator control - allow" is activated, operator control of the objects in the group is permitted.
  Activation of the operator control functions in a cascaded manner:
  - If the "Operator control - allow" property is deactivated, operator control is not possible for any objects.
  - If the "Operator control - allow" property is activated, operator control of objects for which "Operator control - allow" is activated is possible.

  Deactivated objects are displayed grayed out.

See also

Rotating a group and objects in the group (Page 434)

Adding an object property to favorites (Page 377)

4.3.14.13 Adding a property of the group to favorites

Introduction

You can define your own favorite properties in the "Group" screen object.

The following properties are defined as favorites by the system:

- "Name"
- "Top"
- "Left"
- "Height"
- "Width"
- "Visibility"

Adding property to favorites

To add a property to favorites, follow these steps:

1. Right-click on a property that is not defined as a favorite by the system.
2. Select "Add to favorites" in the shortcut menu.
3. To display all favorites, click the icon .

The number of favorite properties is not limited.

Note

You cannot add the aggregated properties of the objects contained in the group to favorites.

Removing property from favorites

To remove a property from favorites, follow these steps:

1. To display all favorites, click the icon .
2. Right-click the favorite property.
3. Select "Remove from favorites" in the shortcut menu.

4.3.14.14 Aggregated properties of the objects in groups

Introduction

The object properties in the group are displayed as aggregated properties in the Inspector window under "Properties > Miscellaneous > Interface". You can either configure these properties with a static value or dynamize them.

Note

Only the properties of the topmost level are available on the interface for aggregated properties. Properties within the sub-hierarchy are not available.

Requirement

The HMI screen containing at least one object group is open.

Aggregated properties with a static value

The following applies for the aggregated properties with a static value:

- When you configure a property with a static value under "Properties > Miscellaneous > Interface", this value is passed to every object in the group that has this property. A similar behavior takes place when a multiple selection is made.
  This static value is not retained at the group level.
- If the static value of a configured property is the same for all objects, the current value is displayed at the group level.
- If the static value of a configured property is different for the objects, "Ambiguous value" is displayed in the input field in the "Static value" column. In this case, too, you can configure any property value in the input field.
- For the aggregated properties at the group level, no validation information is displayed, even if the configured static value of the property is invalid for at least one object.
  Validation only takes place at the level of the individual objects.

Non-aggregatable properties

The following object properties are not displayed under the aggregated properties. You cannot configure or dynamize these object properties at the group level:

- Your own properties of the group, e.g. coordinates or visibility. You can only configure these properties individually.
- Appearance - style item
- Resource list
- Selection item
- Connection status
- Focus - show visual

Static values of the following properties are displayed empty. You cannot configure the properties with a static value.

- Process value
- Text

Dynamization of the aggregated properties

You can configure dynamization of an aggregated property at the group level under "Properties > Miscellaneous > Interface".

As soon as the configured dynamization is triggered in runtime, the dynamization affects all objects in the group that retain this property.

Note

The dynamization type "Flashing" is not available for dynamization of the aggregated properties of a group.

4.3.14.15 Group as part of a multiple selection

Introduction

A group, together with other groups and/or other objects, can be part of a multiple selection. The custom and aggregated properties are displayed under the aggregated properties.

Group as part of a multiple selection

The aggregated group properties are:

- Displayed uncategorized under the "Properties" part.
- Not aggregated with the properties of the objects that are not grouped.

If static property values are defined for multiple selection, the values are passed to all parts of the multiple selection, including custom and aggregated group properties.

If the property is dynamically configured for multiple selection, the dynamic configuration is propagated to all parts of the multiple selection, including custom and aggregated group properties.

Example

If a multiple selection includes the objects of the topmost level as well as groups, you can change the background color of all objects.

1. Specify "Appearance > Background color" for the objects of the topmost level.
2. Specify "Miscellaneous > Interface > Background color" for the grouped objects.

4.3.15 Two-hand operation of operator controls

4.3.15.1 Two-hand operation of operator controls

Introduction

WinCC supports two-hand operation of operator controls for Unified PC. It ensures safe operation of operator controls which are used to change critical system settings, for example, control tags with machine limits.

Locked and unlocked operator controls

You define specific operator controls as "locked operator controls" for two-hand operation of operator controls. Locked operator controls usually cannot be operated in runtime. Operators can only operate the locked operator controls when they press a release button at the same time.

In runtime, locked operator controls can only be accessed with the tab sequence when a release button is pressed at the same time.

Locked operator controls and release buttons

You can configure all operator controls as locked.

You must configure at least one button in the screens as release button. This can be any unlocked button. The unlocking of locked operator controls by pressing the release button has an effect on all open screens.

Display in runtime

The locked operator controls are displayed as dimmed in runtime. The locked operator controls are completely visible when they are unlocked by means of the release button.

Simulation of projects with multi-touch functions

WinCC supports the simulation of configured multi-touch functions. Requirement is that your monitor supports multi-touch operation.

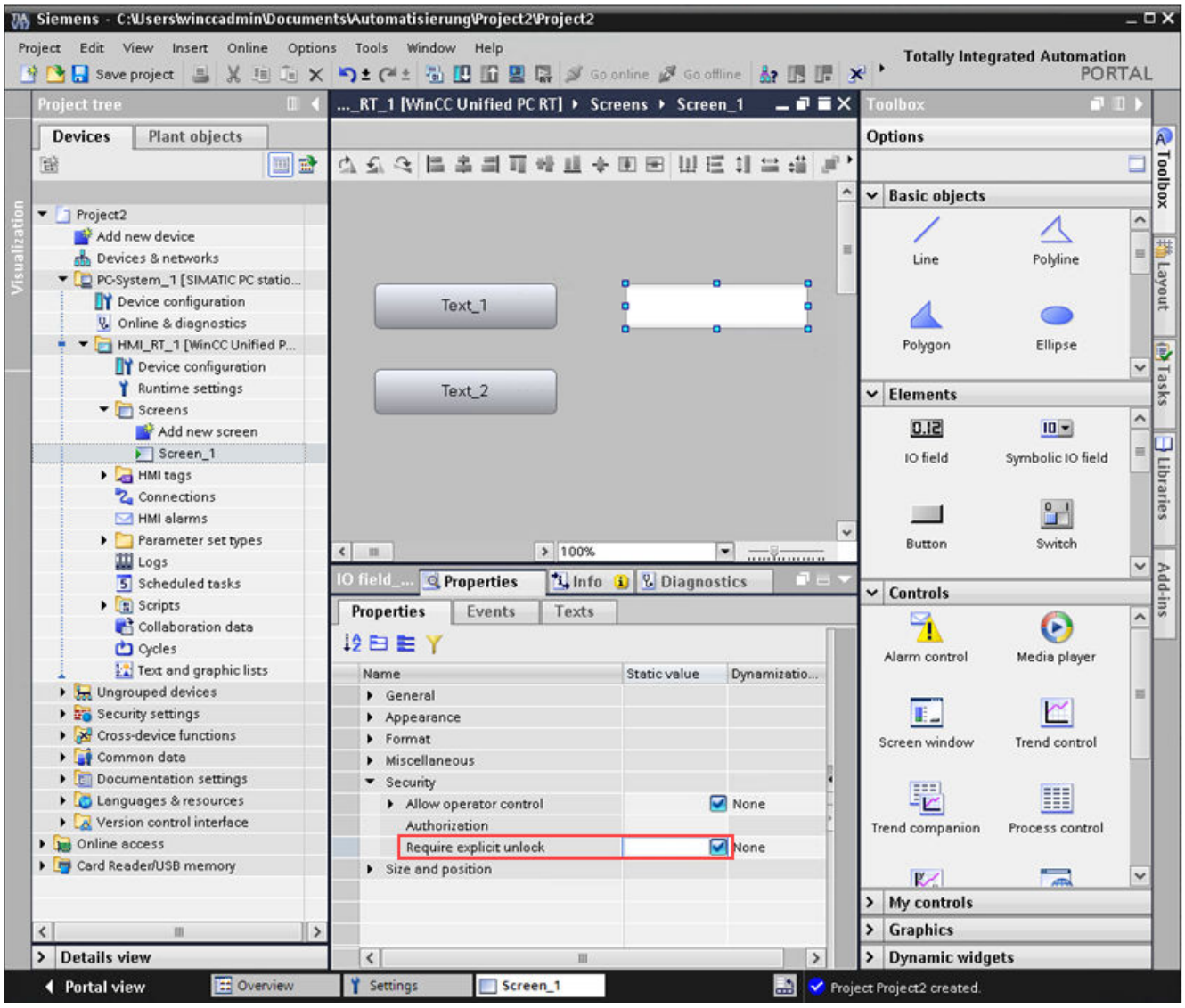
4.3.15.2 Locking and unlocking operator controls

You can lock and unlock operator controls in projects for multi-touch devices. Locked operator controls can only be operated in runtime when the operator presses a release button at the same time.

You can lock and unlock individual operator controls or several operator controls simultaneously.

Procedure

1. Configure operator controls of the type I/O field, button or slider.
2. Select the required operator control(s).
3. To lock the operator controls, enable the "Require explicit unlock" option under "Properties > Properties > Security".



4. To unlock the operator controls, disable the "Require explicit unlock" option under "Properties > Properties > Security".

In runtime, locked operator controls can only be operated when a release button is pressed at the same time.

Note

Locking of operator controls is an add-on to the existing security settings of the operator control. This means that in case of locked operator controls - in addition to pressing the release button - the general operability ("Allow operator control" option) and the required operator control ("Authorization" property) must be present so that the operator control can be operated in runtime.

Defining the release button

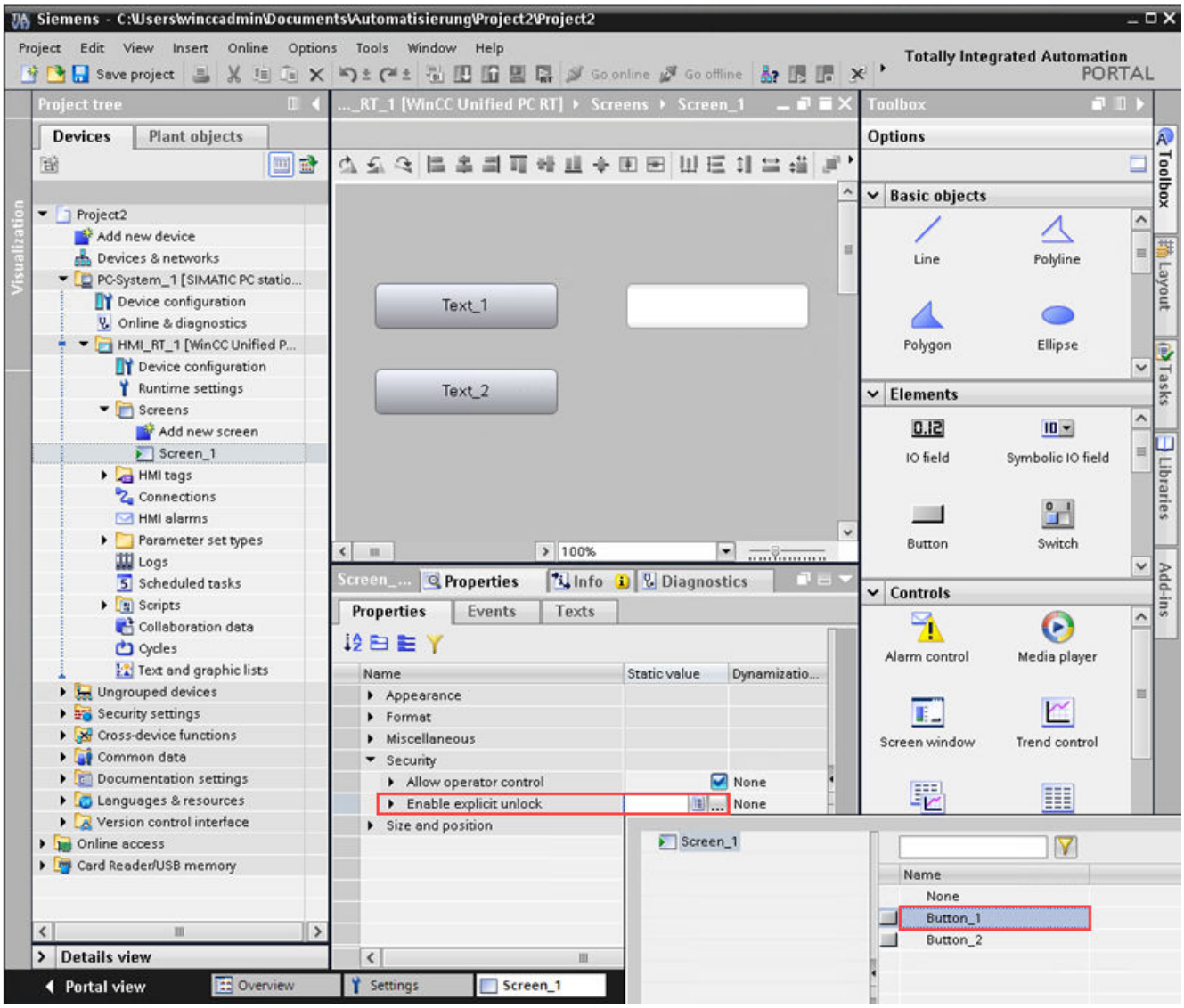
To use the locked operator controls, you must configure at least one release button in one of the displayed screens.

4.3.15.3 Configuring the release button in the screen

So that you can operate locked operator controls on multi-touch devices, configure a release button.

Procedure

1. Select the screen.
2. Select the desired button of the screen under "Properties > Security" under "Enable explicit unlock".



3. To turn a release button back into a normal button, select a different button or "None" under "Properties > Security" under "Enable explicit unlock".

4.4 Configuring text lists and graphics lists

4.4.1 Configuring text lists

4.4.1.1 Basics of text lists

Introduction

Texts are assigned to the values of a tag in a text list. During configuration, you assign the text list to a text field, for example. This supplies the text to be displayed to the object.

You create and edit the text list in the "Text and graphic list" editor. You configure the interface between the text list and a tag at the object that uses the text list.

The availability of the text list is determined by the HMI device used.

Application

You can configure the text list for the following applications:

- Output of texts depending on tag value.
- Display of a selection list in a list box. The associated texts are displayed in the list box depending on the value of the configured tags.

Note

Display of tag values without text

The display of tag values to which no text has been assigned depends on the runtime:

- The display and operating element remains empty.
- Three asterisks *** are displayed.

Ranges for the text list

Three types are available for the text lists:

- Value/Range
  This setting assigns text entries from the text list to integer values or value ranges of a tag. You can select the number of text entries as needed. The maximum number of entries depends on the HMI device you are using.
  You specify a default value which is shown if the value of the tag lies outside the defined range.
- Bit (0, 1)
  This setting assigns text entries from the text list to two states of a binary tag. You can create a text entry for each state of the binary tag.
- Bit number (0-31):
  This setting assigns a text entry from the text list to each bit of a tag. The maximum number of text entries is 32. You use this form of text list, for example, in a sequential control chart when processing a sequencer in which only one bit of the used tag may be set. You influence the behavior of the bit number (0 - 31) with the set bit of the least significance and a default value.

Multilingual texts

You can configure multiple languages for the texts in a text list. The texts will then be displayed in the set language in runtime. To this purpose you set the languages in the Project window under "Languages & Resources > Project languages."

Configuration steps

The following steps are necessary to display texts in a screen object:

1. Creating the text list
2. Assignment of the texts to values or value ranges of a text list
3. Assigning a text list in the display object
4. Assigning a tag

4.4.1.2 Creating a text list

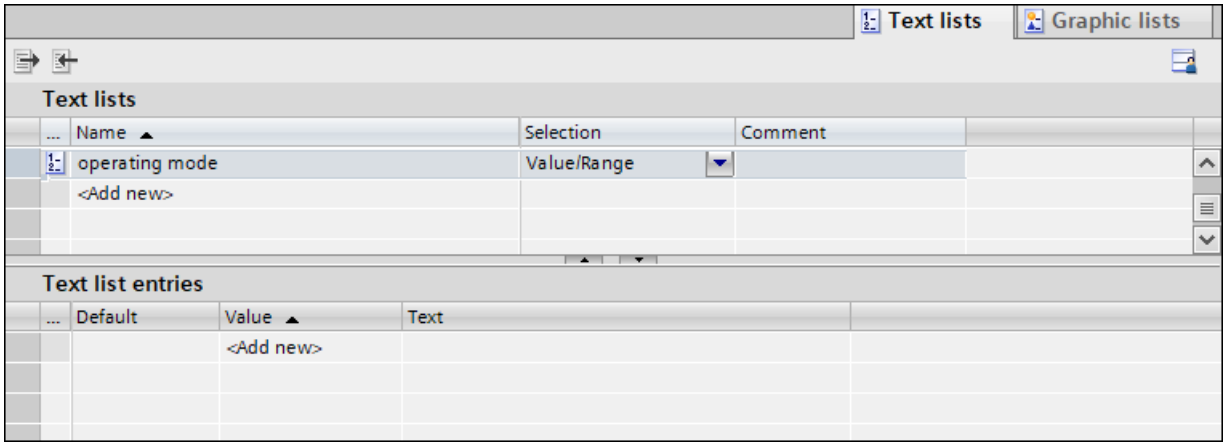
Introduction

The text list allows you to assign specific texts to values and output these in runtime, for example in an I/O field. Specify the I/O field type, for example, as a pure input field.

Procedure

Follow these steps to create a text list:

1. Double-click "Text and graphic lists" in the project tree.
2. Open the "Text lists" tab.



3. Click "Add" in the "Text lists" table. The Inspector window of the text list is open.
4. Assign a name to the text list that indicates its function.
5. Select the text list type under "Selection":
   - Value/Range: Text from the text list is displayed when the tag has a value that lies within the specified range.
   - Bit (0,1): A text from the text list is displayed if the tag has adopted the value 0. Another text from the text list is displayed if the tag has adopted the value 1.
   - Bit number (0-31): Text from the text list is displayed when the tag has the value of the assigned bit number.
6. Enter a comment for the text list.

Result

A text list is created.

4.4.1.3 Assigning texts and values to an area text list

Introduction

For each area text list you specify which texts are displayed at which value range. The entered text is only displayed when the value is within the permitted range.

The following options are available:

- "Range": You enter the minimum value and maximum value for the range.
- "To": You enter the maximum value for the permitted range.

- "Single value": When the specified bit is set, the text is displayed in runtime.
- "From": You enter the minimum value for the permitted range.

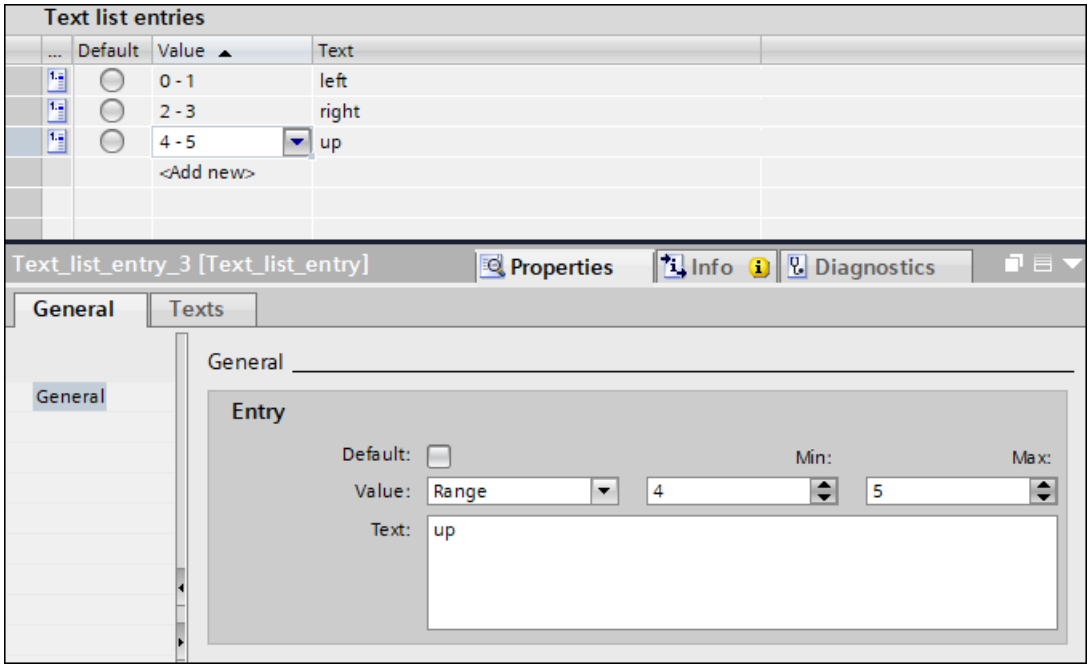
Requirement

- The "Text and graphic list" editor is open.
- The "Text lists" tab is open.
- An area text list has been created and selected.

Procedure

To assign texts and values to a range text list, follow these steps:

1. Click "Add" in the "Text list entries" table.
   The Inspector window for this list entry opens.



2. Select one of the options in the Inspector window "Properties > General > Value" and enter values.
3. Enter the text that is displayed in runtime when the tag has the specified value or lies within the specified range of values under "Text." The text may contain a maximum of 128 characters.
4. Activate the "Default entry" for all unassigned values. The entered text is always displayed when the tag has an undefined value. Only one default entry is possible per list.

   Note

   Selecting the default entry is not possible in runtime.

5. Add additional entries to the text list for additional value ranges.

Result

An area text list is created. Texts that appear in runtime are assigned to the possible values.

4.4.1.4 Assigning texts and values to a bit text list

Introduction

For each bit text list, you specify which text is displayed at which bit value.

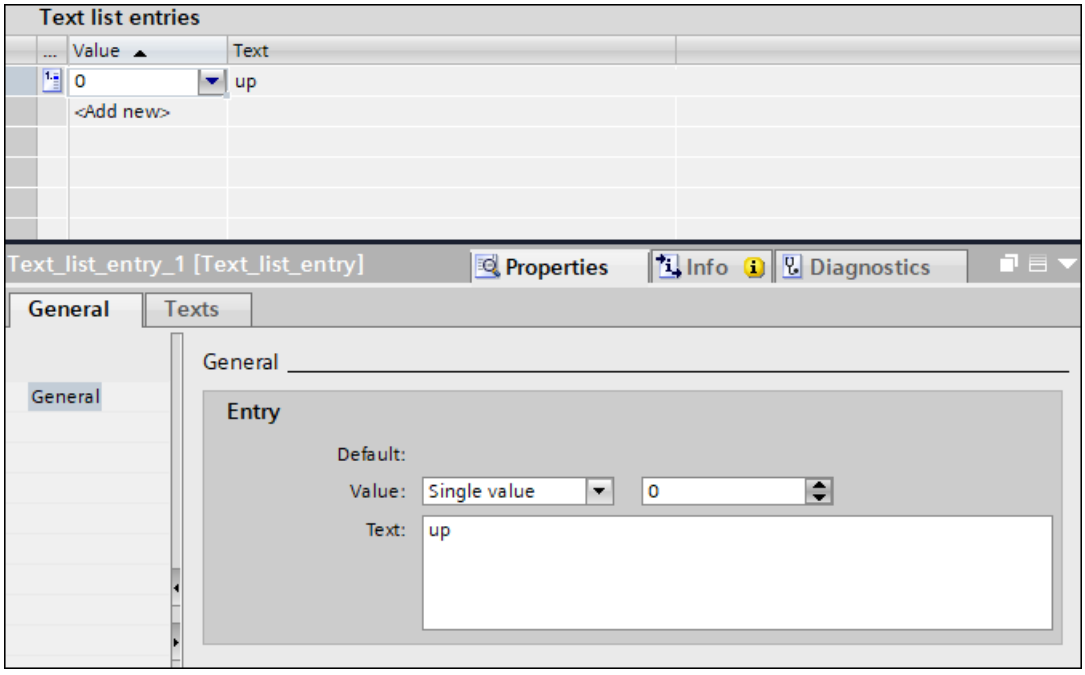
Requirement

- The "Text and graphic list" editor is open.
- The "Text lists" tab is open.
- A bit text list has been created and selected.

Procedure

To assign texts and values to a bit text list, follow these steps:

1. Click "Add" in the "Text list entries" table.
   The Inspector window for this list entry opens.



2. Select the setting "Single value" in the Inspector window "Properties > General > Value" and enter "0" as "value".
3. Under "Text", type in the text that is displayed in runtime when the tag has the value "0". The text may contain a maximum of 128 characters.
4. Click "Add" in the "Text list entries" table. A second list entry is created.

5. Select the setting "Single value" in the Inspector window "Properties > General > Value" and enter "1" as "value".
6. Under "Text", type in the text that is displayed in runtime when the tag has the value "1".

Result

A bit text list is created. Texts that appear in runtime are assigned to the possible values "0" and "1".

4.4.1.5 Assigning texts and values to a bit number text list

Introduction

For each bit number text list you specify which texts are displayed at which bit number.

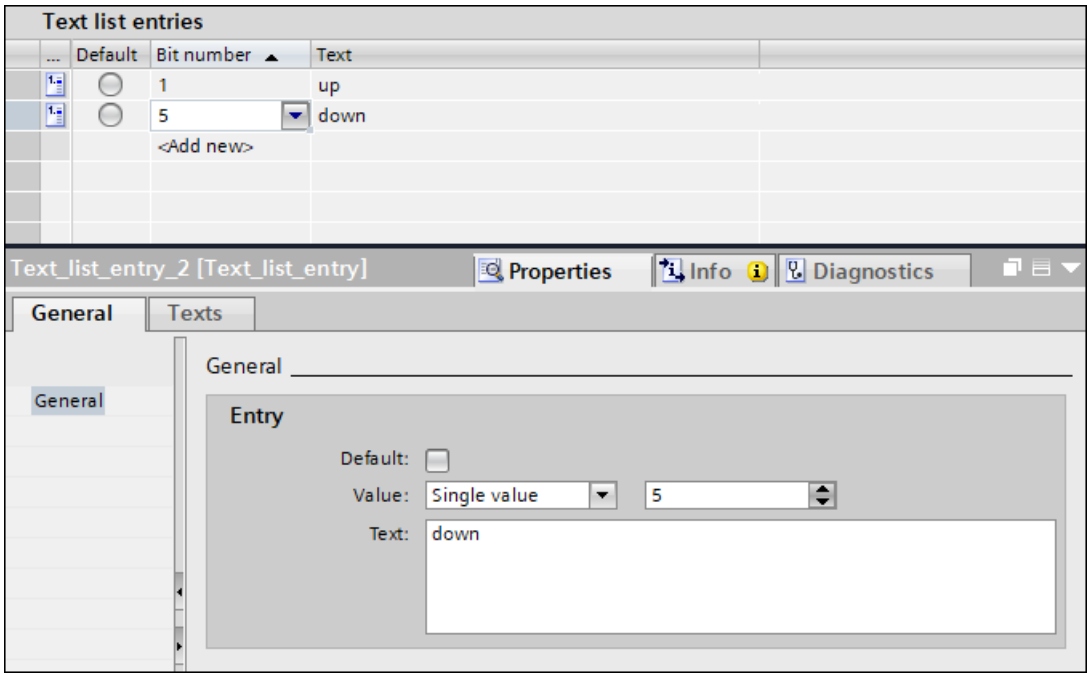
Requirement

- The "Text and graphic list" editor is open.
- The "Text lists" tab is open.
- A bit number text list has been created and selected.

Assign texts and values to the bit number text list

To assign texts and values to a bit number text list, follow these steps:

1. Click "Add" in the "Text list entries" table.
   The Inspector window for this list entry opens.



2. In the Inspector window, select the "Single value" setting under "Properties > General > Value". Enter "5", for example, for "Value".
3. For all unassigned values, enable the "Default" option for the default entry. The text appears when the tag assumes an undefined value. Only one default entry is possible per list.

   Note

   Selecting the default entry is not possible in runtime.

4. Enter the text under "Text". When the tag has taken the value "5", the text is displayed in runtime.
   The text may contain a maximum of 128 characters.
5. You can add additional entries to the text list for more bit numbers.

Result

A bit number text list is created. Texts that appear in runtime are assigned to the specified bit numbers.

Multiline text list entries

For objects with the "Text" property, you can enter the text on multiple lines.

Use the <Shift + Return> key combination to enter a line break in the text entry. Line breaks are represented in the text box by the "¶" paragraph mark.

See also

Entering multiline text (Page 407)

4.4.1.6 Configuring object with a text list

Introduction

The output value and value application for text lists are specified in the display and operating object that displays the texts of the text list in runtime. The properties of these objects are configured as required.

Requirement

- A text list is created. The values have been defined. The texts are assigned to the values.
- You have created a tag.
- The "Screens" editor is open.
- A screen with an object, such as a text box, is open. The text box is selected.

Procedure

1. In the Inspector window under "Properties > Properties > General > Text" in the "Dynamization" column, select the "Resource list" entry.
2. Select the tag whose values determine the display in the text box under "Resource list > Settings > Tag".
3. Select the text list which you want to display in runtime under "Resource list > Settings > Resource list".
4. Click on the button in the "Resource list" line .
   The selected text list opens.
5. Select an entry in the "Text list entries" table as the default entry. The text from the default entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Text" column is displayed in the object.

Tips for an efficient procedure
• Drag-and-drop a text list from the detail view into the screen. A text box is created and linked to the text list. Configure the tags whose values determine the display in the graphic view.

Result

The defined texts of the text list are displayed in the text field in runtime when the tag has the specified value.

See also

Show default entry of text and graphic list in the object (Page 409)

4.4.2 Configuring graphics lists

4.4.2.1 Basics of graphic lists

Introduction

The possible values of a tag are assigned to specific graphics in a graphic list. During configuration, assign the graphic list to a button or a graphic view. This supplies the graphics to be displayed to the object.

The graphic lists are created with the "Text and graphic list" editor. You configure the interface between the graphic list and a tag at the object that uses the graphic list.

The availability of the graphic list is determined by the HMI device used.

Application

You can configure the graphic list for the following situations:

- Selection list with a graphic view.
- State-specific graphic for a button.

Graphic sources

Graphics can be added to the graphic list from the following sources:

- By selecting from the project graphics.
- By selecting an existing file. You can use the following file types: *.bmp, *.ico, *.emf, *.wmf, *.gif, *.tiff, *.png, *.svg, *.jpeg, *.jpg.
- Creating a new file.

Ranges for the graphic list

Three types are available for the graphic lists:

- Value/Range
  This setting assigns graphic entries from the graphic list to integer values or value ranges of a tag. You can select the number of graphic entries as needed. The maximum number of entries depends on the HMI device you are using.
  You specify a default value which is shown if the value of the tag lies outside the defined range.
- Bit (0, 1)
  This setting assigns graphic entries from the graphic list to two states of a binary tag. You can create a graphic entry for each state of the binary tag.
- Bit number (0-31):
  This setting assigns a graphic entry from the graphic list to each bit of a tag. The maximum number of graphic entries is 32. You use this form of graphic list, for example, in a sequential control system when processing a sequencer in which only one bit of the used tags can be set. You influence the behavior of the bit number (0 - 31) with the set bit of the least significance and a default value.

Multilingual graphics

The graphics in a graphic list can be configured as multilingual. The graphics are then be displayed in the set runtime language. To this purpose you set the languages in the Project window under "Languages & Resources > Project languages".

Configuration steps

The following steps are required to display graphics in a graphic view:

1. Creating the graphic list
2. Assignment of the graphics to values or value ranges of a graphic list
3. Assigning a graphic list in the display object.
4. Assigning a tag

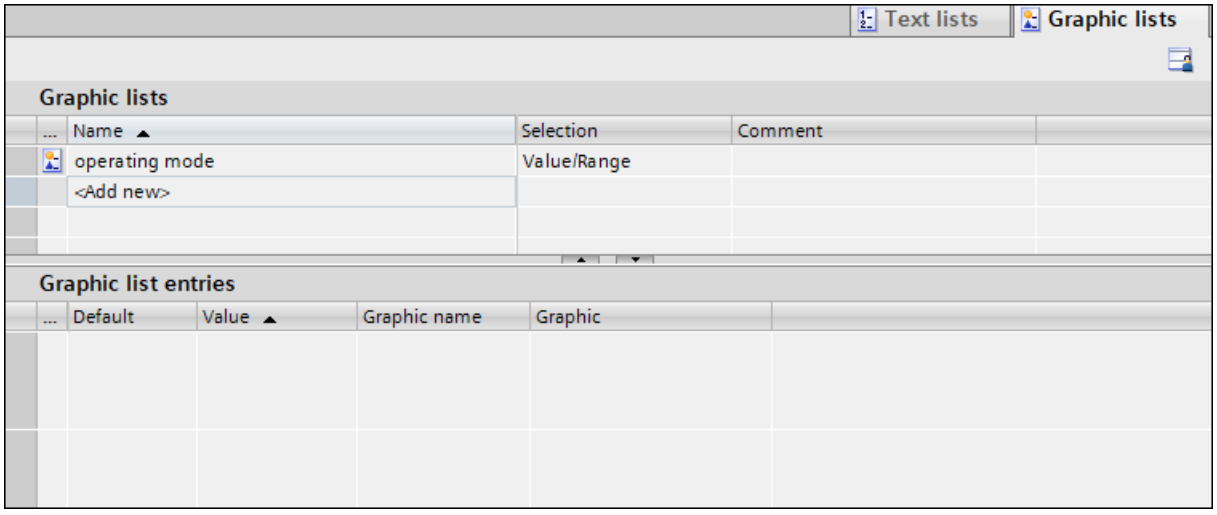
4.4.2.2 Creating a graphic list

Introduction

The graphic list allows you to assign specific graphics to variable values and output these in a graphic I/O field in runtime. Specify the type of the graphic I/O field, for example, as a pure output field.

Procedure

1. Double-click "Text and graphic lists" in the project tree.
2. Open the "Graphic lists" tab.



3. Click "Add" in the "Graphic lists" table. The Inspector window of the graphic list will open up.
4. Assign a name to the graphic list that indicates its function.
5. Select the graphic list type under "Selection":
   - Value/Range: Graphic from the graphic list is displayed when the tag has a value that lies within the specified range.
   - Bit (0,1): A graphic from the graphic list is displayed when the tag has the value 0. A different graphic from the graphic list is displayed when the tag has the value 1.
   - Bit number (0-31): Graphic from the graphic list is displayed when the tag has the value of the assigned bit number.
6. Enter a comment for the graphic list.

Result

A graphic list is created.

4.4.2.3 Assigning graphics and values to an area graphic list

Introduction

For each area graphic list you specify which graphics are displayed at which value range. The selected graphic is only displayed when the value is within the permitted range.

The following options are available:

- "Range": You enter the minimum value and maximum value for the range.
- "To": You enter the maximum value for the permitted range.

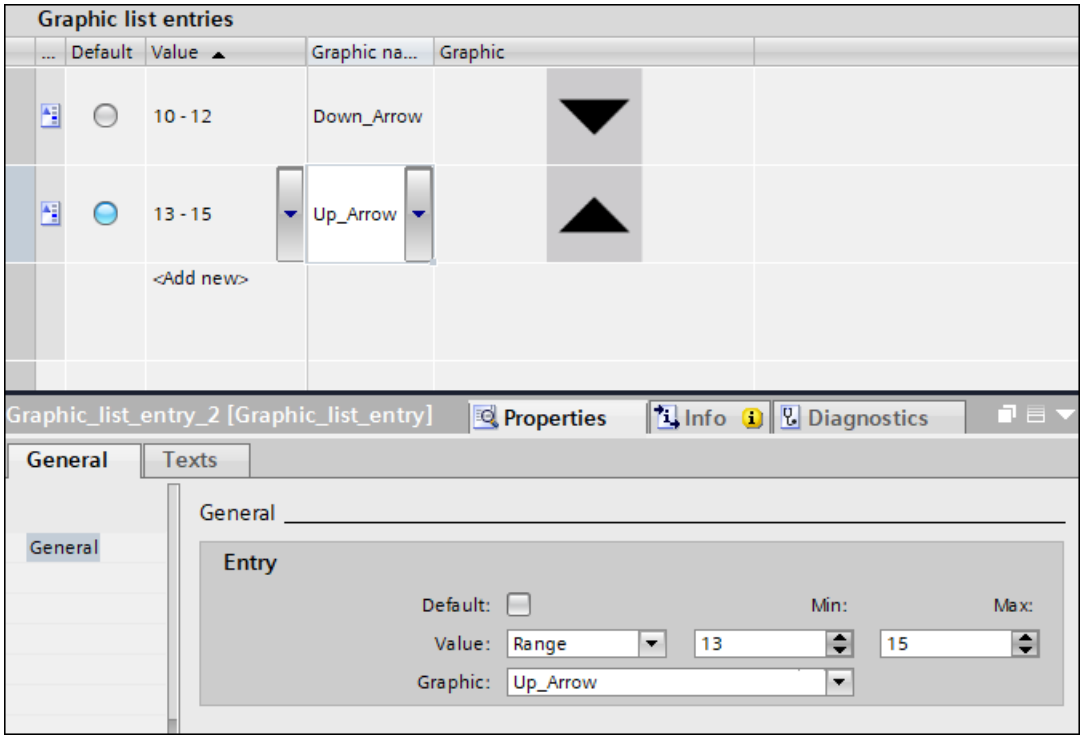
- "Single value": When the specified bit is set, the selected graphic is displayed in runtime.
- "From": You enter the minimum value for the permitted range.

Requirement

- The "Text and graphic list" editor is open.
- The "Graphic list" tab is open.
- An area graphic list has been created and selected.

Procedure

1. Click "Add" in the "Graphic list entries" table.
   The Inspector window for this list entry opens.



2. Select an option in the Inspector window "Properties > General > Value". Enter values.
3. In the "Graphic" column, select a graphic to be displayed in runtime if the tag has the specified value or is within the specified value range.
4. For all unassigned values, enable the "Default" option for the default entry. The graphic is displayed when the tag has an undefined value. Only one default entry is possible per list.

   Note

   Selecting the default entry is not possible in runtime.

5. Add additional entries to the graphic list for additional value ranges.

	**Tips for an efficient procedure**
Inserting a graphic using drag-and-drop operation: 1. Select a graphic in the library or in your file system. 2. Drag-and-drop the graphic into the "Graphic list entries > Graphic" table.	

Result

An area graphic list is created. Graphics that appear in runtime are assigned to the possible values.

4.4.2.4 Assigning graphics and values to a bit graphic list

Introduction

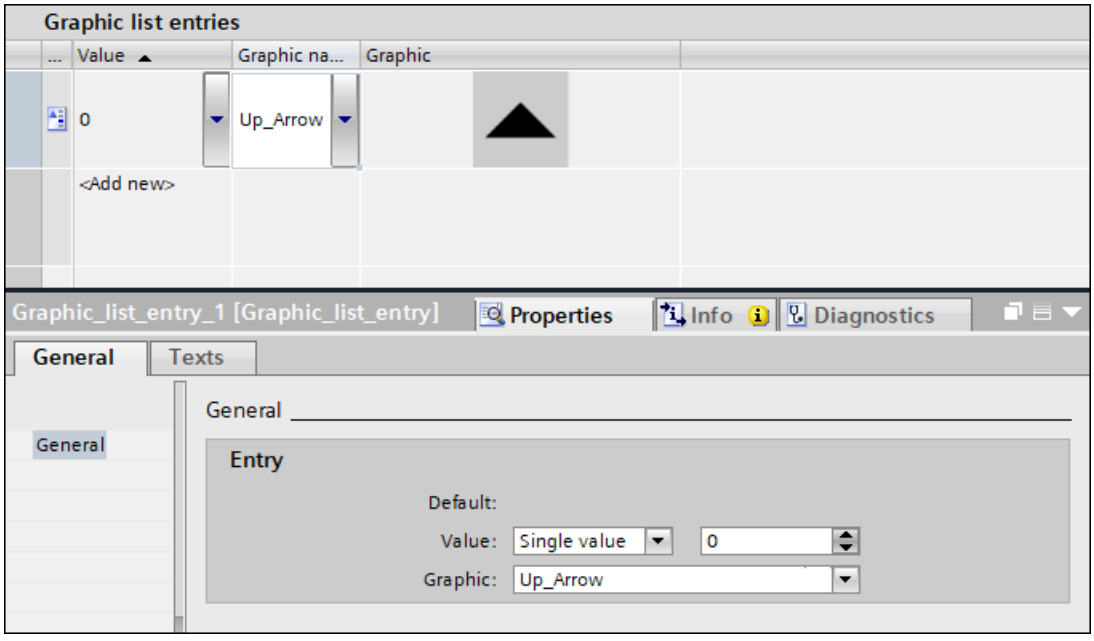
For each bit graphic list you specify which graphic is displayed at which bit value.

Requirement

- The "Text and graphic list" editor is open.
- The "Graphic list" tab is opened.
- A bit graphic list has been created and selected.

Procedure

1. Click "Add" in the "Graphic list entries" table.
   The Inspector window for this list entry opens.



2. Select the setting "Single value" in the Inspector window under "Properties > General > Value" and enter "0" as "value".
3. Click "Add" in the "Graphic list entries" table. A second list entry is created.
4. Select the setting "Single value" in the Inspector window under "Properties > General > Value" and enter "1" as "value".
5. Select a graphic that is displayed in runtime when the tag has the value "1".

	Tips for an efficient procedure
Inserting a graphic using drag-and-drop operation: 1. Select a graphic in the library or in your file system. 2. Drag-and-drop the graphic into the "Graphic list entries > Graphic" table.	

Result

A bit graphic list is created. Graphics that appear in runtime are assigned to the values "0" and "1".

4.4.2.5 Assigning graphics and values to a bit number graphic list

Introduction

For each bit number graphic list you specify which graphics are displayed at which bit number.

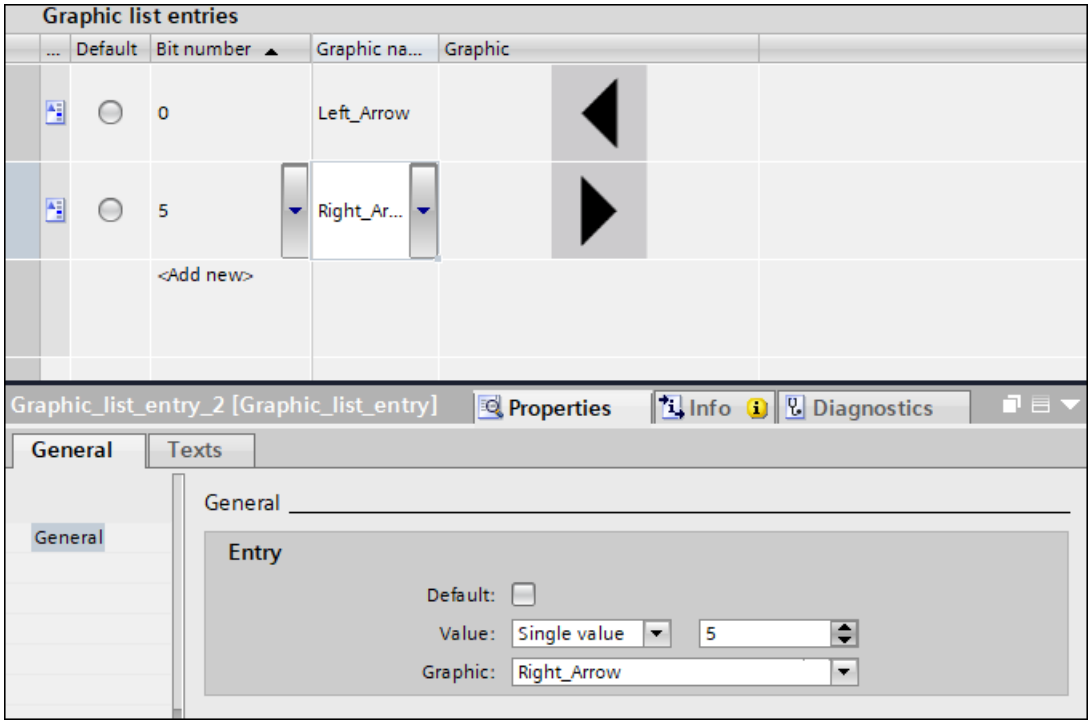
Requirement

- The "Text and graphic list" editor is open.
- The "Graphic list" tab is open.
- A bit number graphic list has been created and selected.

Assigning graphics and values to the bit number graphic list

To assign the graphics and values to a bit number graphic list, follow these steps:

1. Click "Add" in the "Graphic list entries" table.
   The Inspector window for this list entry opens.



2. Select the "Single value" settings in the Inspector window "Properties > General > Value". Enter "5", for example, for "Value".
3. For all unassigned values, enable the "Default" option for the default entry. The graphic is displayed when the tag has an undefined value. Only one default entry is possible per list.

   Note

   Selecting the default entry is not possible in runtime.

4. Enter the graphic under "Properties > General > Graphic". When the tag has taken the value "5", the graph is displayed in runtime.
5. You can add additional entries to the graphic list for more bit numbers.

	Tips for an efficient procedure
Inserting a graphic using drag-and-drop operation: 1. Select a graphic in the library or in your file system. 2. Drag-and-drop the graphic into the "Graphic list entries > Graphic" table.	

Result

A bit number graphic list is created. Graphics that appear in runtime are assigned to the specified bit numbers.

4.4.2.6 Configuring objects with a graphic list

Introduction

The output value and value application for graphic list are specified in the display and operating object that displays the graphics of the graphic list in runtime. The properties of these objects are configured as required.

Requirement

- A graphic list is created. The values have been defined. Graphics have been assigned to the values.
- You have created a tag.
- The "Screens" editor is open.
- A screen with an object, such as a graphic view, is open. The graphic display is selected.

Procedure

1. In the Inspector window under "Properties > Properties > General > Graphics", in the "Dynamization" column, select the "Resource list" entry.
2. Under "Resource list > Settings > Tag", select the tag whose values determine the display in the graphic view.
3. Select the graphic list which you want to have displayed in runtime under "Resource list > Settings > Resource list".
4. Click on the button in the "Resource list" line .
   The selected graphic list opens.
5. Select an entry in the "Graphic list entries" table as the default entry. The graphic from the standard entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Graphic" column is displayed in the object.

Tips for an efficient procedure

- Drag a graphics list from the detail view of the text and graphics lists to the screen. A graphic view is created and linked to the graphics list. Configure the tags whose values determine the display in the graphic view.

Result

If the tag adopts the specified value, the defined graphics are displayed in the graphic view in runtime.

4.5 Configuring dynamization

4.5.1 Basics of dynamizing screens

Dynamizing objects

Dynamics are used to change the properties of screen objects and screens in runtime depending on another value. The source for this value changes is referred to as "Dynamization type".

Dynamization types

The following table shows the dynamization types available in WinCC:

Dynamization type	Description	Supported property classes	Examples
Tag	Defines the property value depending on the tag value.	All	"Process value" or "Left" properties
Script	Defines the property value depending on the return value.	All	"Process value" or "Left" properties
Resource list	Defines the property value depending on an entry from a text list or graphic list.	Text / Graphic	Properties "Text", "Tooltip" or "Graphic".
Flashing	Defines that the property flashes in configurable colors.	Colors	Properties "Foreground color" or "Border color".
Expression	Specifies the property value depending on several tag values. The tag values are linked by logical operators.	Depends on the screen object	"Size" or "Background color" properties

Examples of dynamizations

The table below shows typical application examples for each type of dynamization:

Dynamization type	Application example
Tag	Visualize the level. The "Process value" property of a bar graph is dynamized with a tag that contains the level from the PLC.
Script	Simulate the filling process. To simulate a movement of bottles on a conveyor belt, the properties "Left", "Top" and "Visible" are dynamized with scripts.
Resource list	Display the plant status. The meaning of a quality code is saved in a text list. Depending on the transferred numerical quality code, its meaning is displayed on the HMI device.
Flashing	Visualize limit violations. When the level of a tank drops below a limit, the visualized tank is to flash in two signal colors.
Expression	Visualization of machine states. Machine states of different plant objects are stored in tags of the type "Bool". Complex visualizations are derived from the combination of tag values.

See also

Automatically filling in of property values for an object collection (Page 379)

4.5.2 Displaying dynamization of the properties

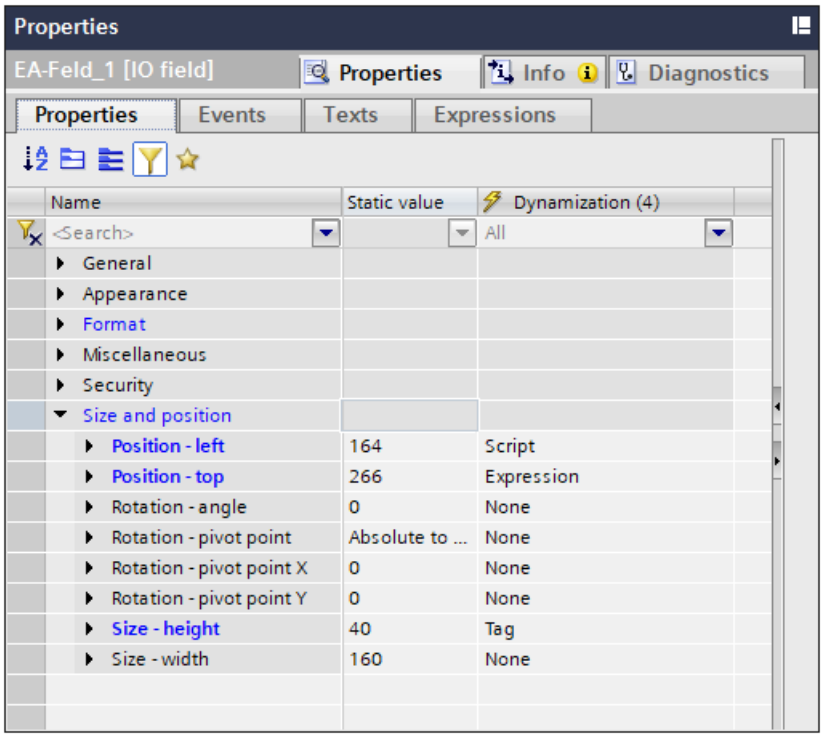
Introduction

You can see in the Inspector window of an object which object properties have been dynamized.

Displaying dynamization of the property

Dynamized properties are shown in dark blue font and in bold in the Inspector window in the "Name" column.

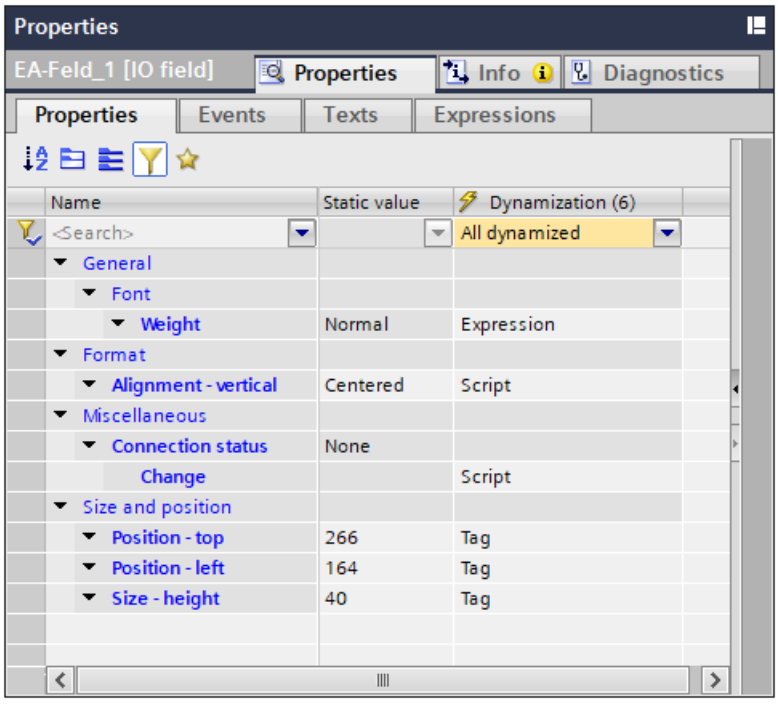
Groups containing a dynamized property are shown in dark blue font.



Filter dynamized properties

If you use the "Filter" function and select "All dynamized" in the "Dynamization" column, the following is displayed:

- All dynamized properties in dark blue font and bold.
- The groups in which a property is dynamized in dark blue font.



4.5.3 Find type of dynamization

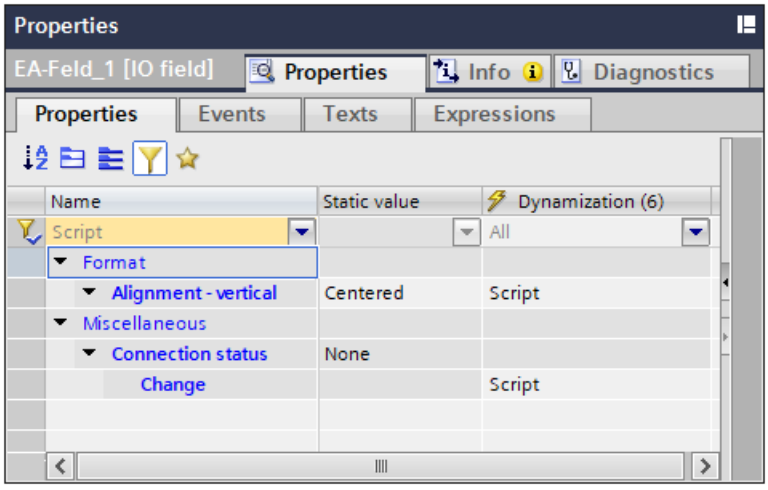
Introduction

You can search for the individual type of dynamization using the "Filter" icon .

Find type of dynamization by input

To search for a type of dynamization by entering the type of dynamization, follow these steps:

1. Click on the "Filter" icon.
2. Enter a type of dynamization.
3. Confirm your selection with the <Enter> key.

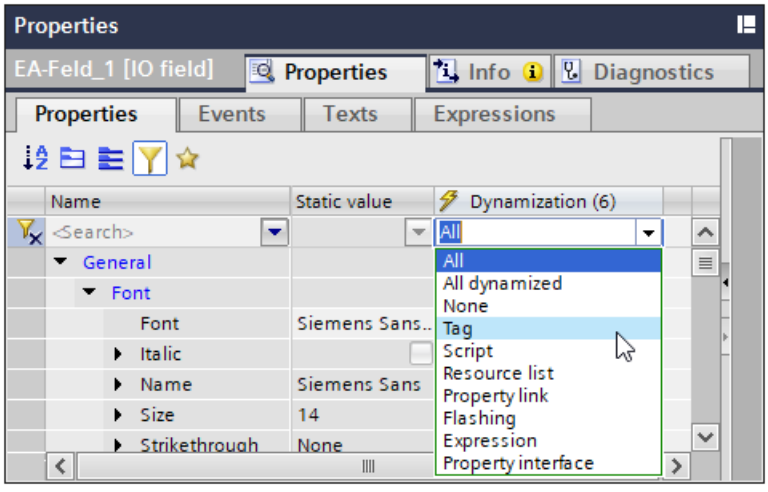


All properties with the selected type of dynamization are displayed.

Find type of dynamization by selection

To search for a type of dynamization by selecting the type of dynamization, follow these steps:

1. Click on the "Filter" icon.
2. Click the arrow in the first cell of the "Dynamization" column.
3. Select the desired type of dynamization from the drop-down list.



All properties with the selected type of dynamization are displayed.

4.5.4 Changing a dynamization for multiple objects

Introduction

You can change the dynamization of a property for several objects at the same time.

Changing a dynamization for multiple objects

To change the dynamization of a property for several objects at the same time, follow these steps:

1. Select several objects in the screen via multiple selection.
2. Select a property.
3. Dynamize the property, for example, by using a script.

The dynamization of the property is applied to all selected objects that have the property.

See also

Select multiple objects (Page 361)

4.5.5 Dynamizing object properties

4.5.5.1 Dynamizing an object property with a tag

Dynamizing an object property with a tag

Introduction

When you dynamize an object property with a tag, the object property is changed in runtime depending on the tag value.

Requirement

- A screen is open.
- An object is configured.
- You have configured a tag.
- The object property supports the dynamization type "Tag".

Procedure

To dynamize an object property using a tag, follow these steps:

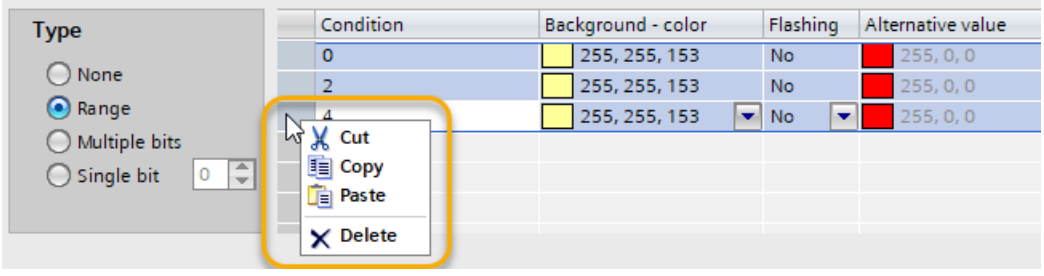
1. Select the object.
2. Under "Properties > Properties > Dynamization" select the object property in the Inspector window.
3. Select the "Tag" option.
4. Select the tag. Only the relevant tags are displayed in the selection dialog. If you want to see all tags, enable "Show all" in the selection dialog.
5. If required, enable the options "Indirect addressing" or "Read only" in the "Settings" area.
6. Define the type of tag:
   - None
   - Range
   - Multiple bits
   - Single bit
7. Define the properties for the respective type of tag.

The object property tag is dynamized with a tag. The tag value specifies the property value in runtime.

Transferring property conditions

With "Copy" and "Paste" you can transfer the conditions of an object property to the same or similar properties of another object:

1. Select one or more conditions in the "Tag > Type" table.
2. Select "Copy" in the shortcut menu.



3. Select a different object or a different object property in the screen editor.
4. Paste the copied conditions into the "Tag > Type" table.

Note the following:

- Any existing conditions are overwritten on pasting.
- The data types must be the same or similar, e.g. "Background - color" and "Background - alternative color".

- Inserting data is possible even if no tag is assigned to the property.
- When pasting data, if the number of copied conditions is greater than the possible number of conditions at the target object, only the maximum possible number of conditions will be pasted.

See also

Automatically filling in of property values for an object collection (Page 379)

Dynamizing an object property with a tag of the "Range" type

Introduction

When you dynamize an object property with a tag, the object property is changed in runtime depending on the tag value.

The following example shows the dynamization of a button. The button changes its alternative background color depending on the tag of the "Range" type.

Requirement

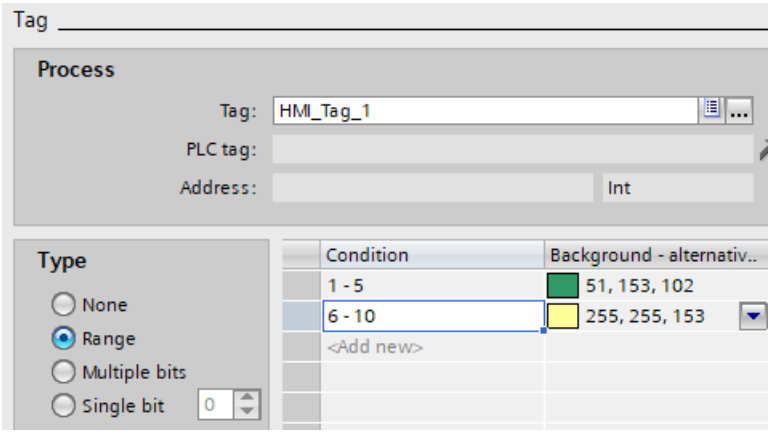
- A screen is open.
- An object, for example a button, is configured.
- You have configured a tag.
- The object property supports the dynamization type "Tag".

Selecting a tag of the "Range" type

To dynamize an object property with a tag of the "Range" type, follow these steps:

1. Select the object, for example, a button.
2. In the Inspector window under "Properties > Properties > General > Text > Background - Alternative color", dynamize an object property, such as "Background - Alternative color".
3. Select "Tag" in the "Dynamization" column.
4. Select a tag in the "Tag > Process" dialog.
5. Select "Range" as type".

6. Enter the ranges in the "Condition" column.
7. Define the alternative background color for each area in the "Background - Alternative Color" column.



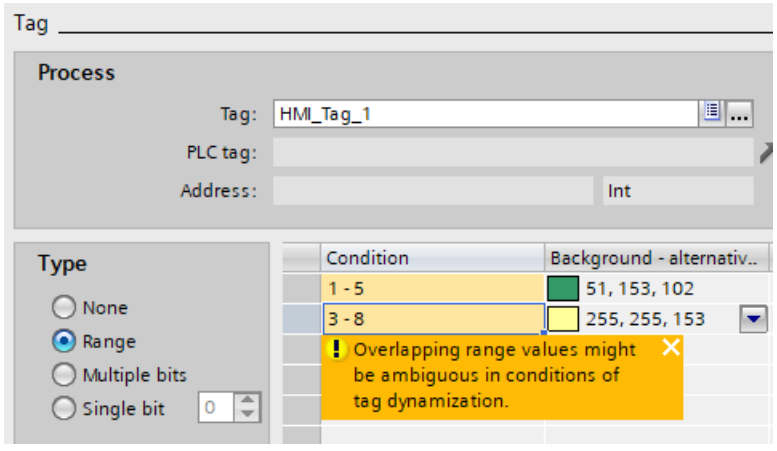
Result

The color of the button is changed accordingly depending on the position of the tag in the value range in runtime.

Overlapping ranges

If there is overlapping of values in the individual ranges, note that:

- Cells where range overlaps occur are highlighted in color.



- The overlaps are reported as a warning when compiling the project. Downloading to the device is possible.
- The first highlighted value is always used in runtime.

See also

Automatically fill in property values for tags (Page 472)

Dynamizing an object property with a tag of the "Multiple bits" type

Introduction

When you dynamize an object property with a tag, the object property is changed in runtime depending on the tag value.

The following example shows the dynamization of a button. The button changes its alternative background color depending on the tag of the "Multiple bits" type.

Requirement

- A screen is open.
- An object, for example a button, is configured.
- You have configured a tag.
- The object property supports the dynamization type "Tag".

Selecting a tag of the "Multiple bits" type

To dynamize an object property with a tag of the "Multiple bits" type, follow these steps:

1. Select the object, for example, a button.
2. In the Inspector window under "Properties > Properties > General > Text > Background - Alternative color", dynamize an object property, such as "Background - Alternative color".
3. Select "Tag" in the "Dynamization" column.
4. Select a tag in the "Tag > Process" dialog.
5. Select "Multiple bits" as the type.
6. Enter the bit number.
7. Define the alternative background color for each bit number in the "Background - Alternative color" column.

Result

When the tag has accepted the assigned bit number in runtime, the color of the button is changed accordingly.

See also

Automatically fill in property values for tags (Page 472)

Dynamizing an object property with a tag of the "Single bit" type

Introduction

When you dynamize an object property with a tag, the object property is changed in Runtime depending on the tag value.

The following example shows the dynamization of a button. The button changes its alternative background color depending on the tag of the "Single bit" type.

Requirement

- A screen is open.
- An object, for example a button, is configured.
- You have configured a tag.
- The object property supports the dynamization type "Tag".

Selecting a tag of the "Single bit" type

To make an object property dynamic with a tag of the "Single bit" type, follow these steps:

1. Select the object, for example, a button.
2. In the Inspector window under "Properties > Properties > General > Text > Background - Alternative color", dynamize an object property, such as "Background - Alternative color".
3. Select "Tag" in the "Dynamization" column.
4. Select a tag in the "Tag > Process" dialog.
5. Select "Single bit" as the type.
6. Select the bit position, for example, "3".
7. Define the properties for the "0" and "1" value of the condition.

Result

When the tag at "3" position has taken the value "1" in runtime, the color of the button is changed.

Automatically fill in property values for tags

Introduction

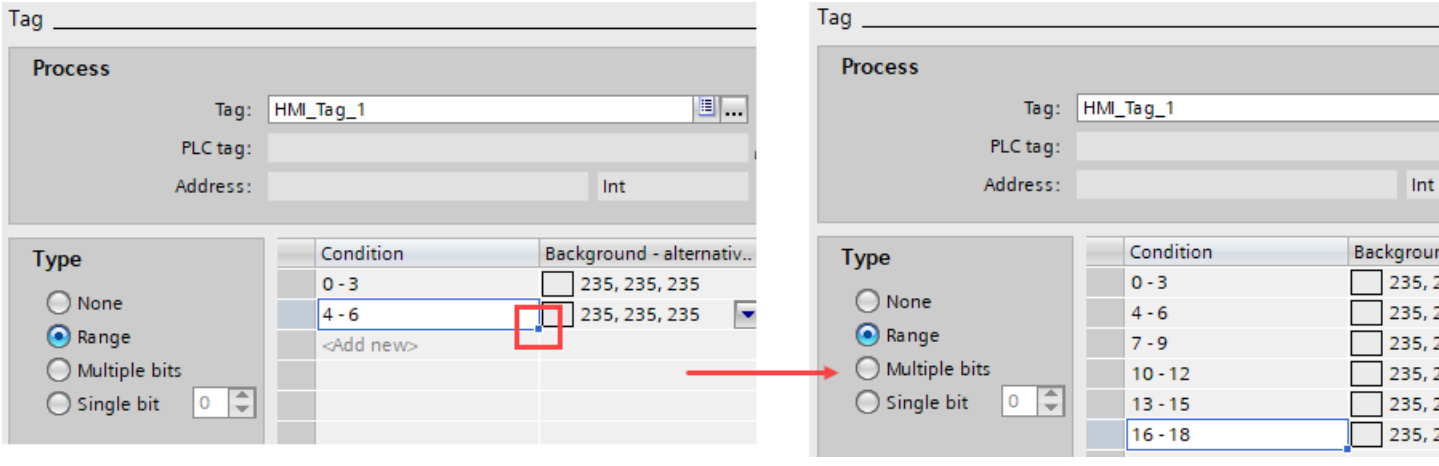
You can automatically fill in the properties of tags of the "Range" and "Multiple bits" types.

Automatically fill in property values for tags

To fill in the properties automatically, do the following:

1. Select a cell in the "Tag" dialog in the right part of the Inspector window.
2. Drag the blue border up or down. The value is applied to the target cells.

If you select multiple cells in the "Condition" column and there is a logical relationship between the values, the values of the destination cells are adapted according to the relationship.



See also

Dynamizing an object property with a tag of the "Range" type (Page 469)

Dynamizing an object property with a tag of the "Multiple bits" type (Page 471)

4.5.5.2 Dynamizing an object property with a script

Introduction

Object properties can be dynamized by scripts. The execution of the script is started by a trigger. A cycle or a tag is set as the trigger.

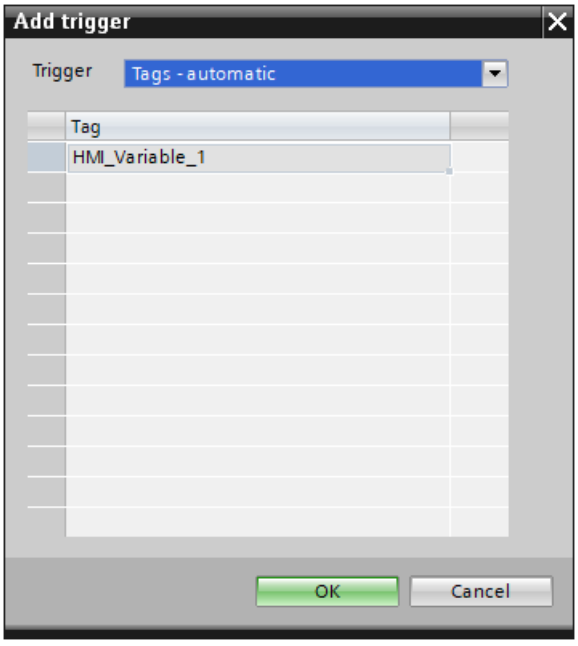
Requirement

- A screen is open.
- An object is configured.
- An object property supports the dynamization type "Script".

Procedure

To dynamize an object property using a "Script", follow these steps:

1. Select the object.
2. Dynamize the object property, e.g. "Background - color", under "Properties > Properties > Dynamization" in the Inspector window:
   - Select a dynamized property.
   - Select the option "Script".
     The editor for scripts is displayed.
   - If necessary, create a "Global definition".
     Click "Global definition".
     Write the code for the global definition.
   - Write the code for the script.
   - Insert a tag into the script.
     Referenced tags are automatically offered as triggers.
3. Select the trigger that triggers the dynamization in Runtime.
   The dialog for selecting a trigger is displayed.



Select

- a tag that is referenced in the script automatically (default)
  - or -
- a tag that is not referenced in the script
  - or -
- a cycle.

Result

The object property is dynamized with a script. The return value of the script specifies the property value in Runtime.

Note

The dynamization of an event is only monitored regarding an operator authorization if the triggering event, e.g. "Press button", is triggered by a user.

4.5.5.3 Dynamizing an object property with a resource list

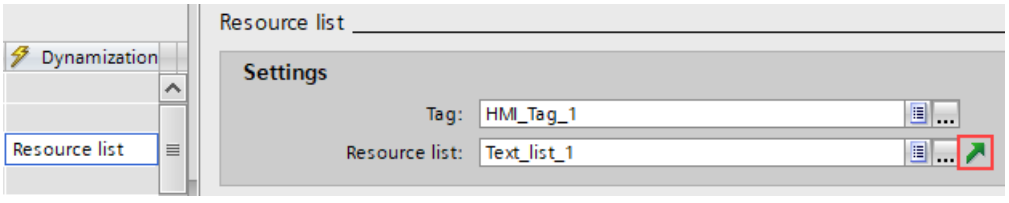
Requirement

- A screen is open.
- An object is configured.
- You have configured a tag.
- A text list or graphic list is configured.
- The object property supports the "Resource list" dynamization type.

Procedure

To dynamize an object property using a "Resource list", follow these steps:

1. Select the object.
2. In the Inspector window, select "Properties > Properties > General > Text".
3. Select "Resource list" in the "Dynamization" column.
4. Under "Settings > Resource list" in the "Resource list" dialog, select a text list, for example.
5. Click on the button in the "Resource list" line .
   The selected text list is opened directly for editing.



6. Select an entry in the "Text list entries" table as the default entry. The text from the default entry is displayed in the object.
   If you have not specified a default entry, the first entry in the "Text" column is displayed in the object.
   The same applies for a graphic list.

Result

The object property tag is dynamized with a resource list. The tag value specifies the entry from the configured text list or graphic list that is displayed in runtime.

See also

Show default entry of text and graphic list in the object (Page 409)

4.5.5.4 Dynamizing an object property with flashing

Introduction

You can display objects as flashing in runtime. You can configure the flashing characteristics for each color setting of an object that supports flashing in the Inspector window and select the colors, the condition, and the flash rate.

Note

Flashing in runtime does not change the color value of the property.

Requirement

- A screen is open.
- An object is configured.
- The object property supports the dynamization type "Flashing".

Dynamizing an object property with flashing

To dynamize an object property with flashing, follow these steps:

1. Select the object.
2. In the Inspector window under "Properties", select the property for which you want to define the flashing characteristics, for example, "Background - color".
3. Select "Flashing" in the "Dynamization" column. The "Flashing" page appears.
4. Select the flash colors. Flashing is only visible in runtime when there is a difference between the two colors.
5. Select the condition for the object flashing in runtime.
6. Select the flash rate.

The object property is dynamized with flashing using the dynamization type "Flashing". When the configured condition occurs in runtime, the object property flashes in the configured colors and at the set rate.

Condition and rate

The following options are available for the condition:

- "Never": You disable the flashing.
- "Always": You enable the flashing.
- "Range violation": The property flashes when the configured permissible range is violated.

Note

Flashing for a range violation only works if you have linked the object to a PLC tag. When the value of the PLC tag lies outside the defined range, the object will start flashing automatically.

The following options are available for the rate:

- "Slow"
- "Medium"
- "Fast"

Result

The object property is dynamized. When the configured condition occurs in runtime, the object property flashes in the configured colors and at the set rate.

4.5.5.5 Dynamization by expressions

Introduction

Properties can be dynamized depending on several tags. In an expression, tags are linked by logical operators. The result of the expression determines the dynamization. It is possible, for example, to link two tags of the type "Bool" with the operator AND. If both tags return the value TRUE, then the result of the expression is also TRUE and the property value is set accordingly.

The following applies to the input values for logical operations:

- Value 0 = FALSE
- Value not equal to zero = TRUE

Requirement

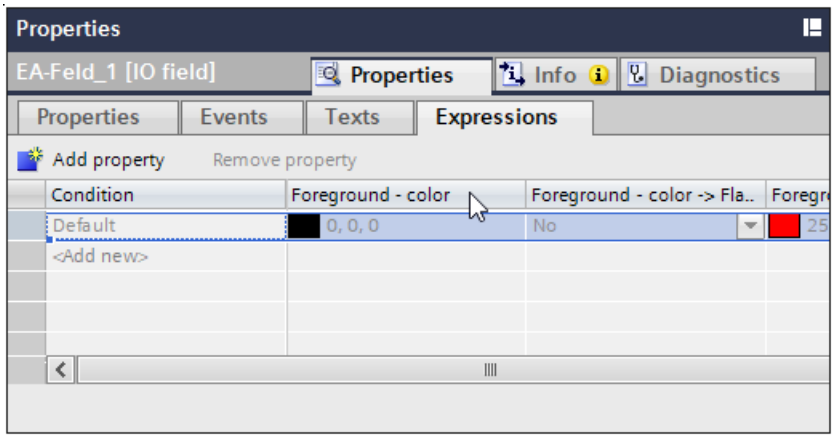
- A screen is open.
- An object is configured.

Dynamizing an object property with an expression

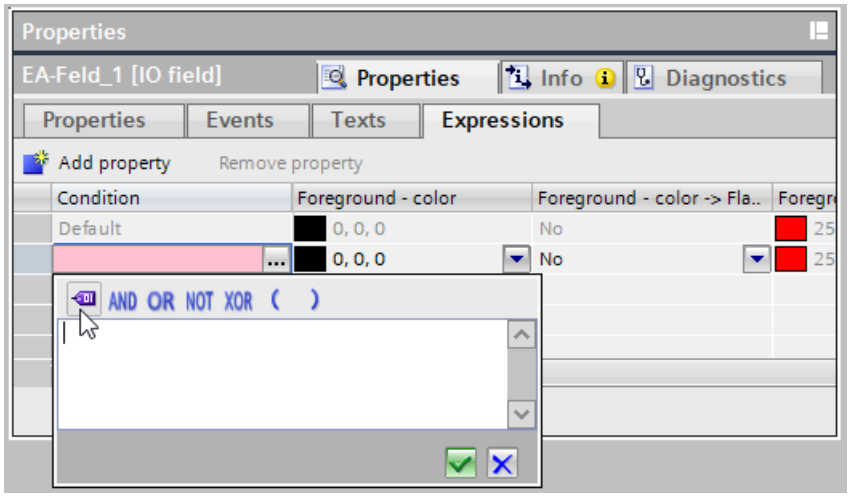
To dynamize an object property with an expression, follow these steps:

1. Select the object.
2. In the Inspector window, under "Properties", select the "Expressions" tab.

3. Select "Add property".
   This selects a property of the screen object to be dynamized with an expression.
   The property is displayed with the default values.



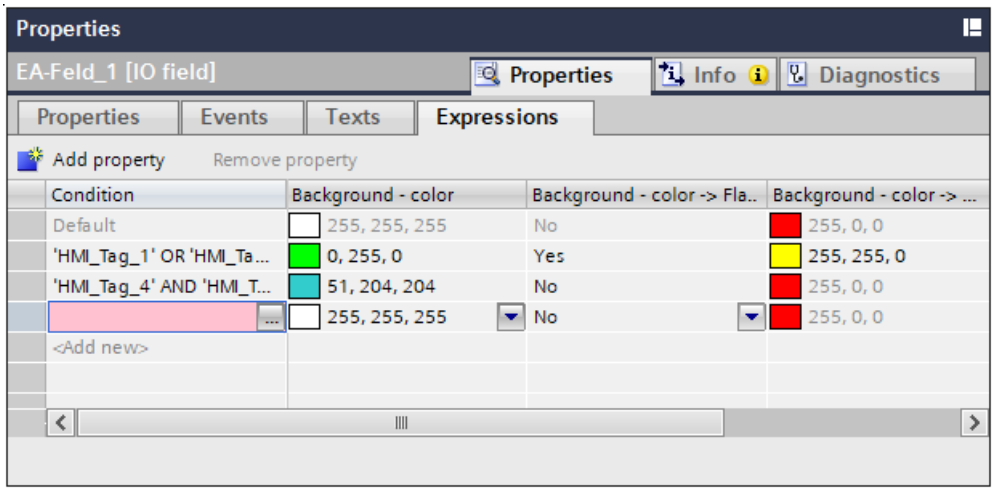
4. To specify dynamization with an expression for the property, click on "<Add new>" in the "Condition" column.
5. Click ... .
   The editor for expressions is displayed.



   - To insert a tag, click the tag icon.
     The selection dialog for tags is displayed.
   - To insert an operator at the cursor position, click AND, OR, NOT or XOR.
   - To insert a bracket at the cursor position, click on the symbols for brackets.

6. To save the expression, click .
7. Determine the properties to apply to the screen object when the expression returns the TRUE value.

8. If necessary, add further conditions for the currently selected property.



9. To dynamize another property of the selected object with an expression, click "Add property". Properties for which a dynamization has already been configured cannot be added a second time.

Result

The expression is evaluated when one of the tag values changes.

Properties change as soon as the result of the expression changes.

If none of the conditions returns TRUE, the default value is assumed.

If multiple conditions are defined, the first condition in the list that returns TRUE is applied.

Expressions that cannot be evaluated are skipped, e.g. because of a syntax error or a tag that cannot be accessed.

4.5.5.6 Examples

Dynamizing a graphic property with a tag

You can dynamize a property containing a graphic with a tag.

The following object properties can contain a graphic:

- Graphic (e.g. a list box)
- Graphic - pressed button (e.g. a button)
- Background graphic (one screen only)
- Marker graphic (e.g. an alarm control)
- Icon (e.g. a trend control)

Task

In this example, you can learn how to dynamize the "Graphic" property at the "Button" object with a tag of the type "Area".

You configure:

- Button
- HMI tag "MyTag"
- Graphic in the project graphics

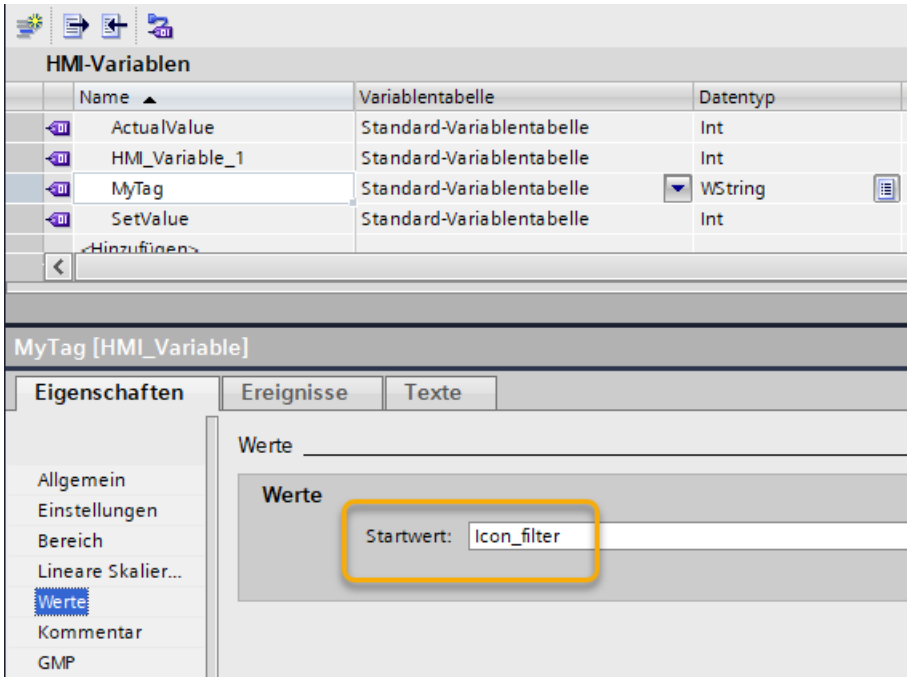
Requirement

- A project is open.
- A screen is configured.
- A button is configured in the screen.

Adding a graphic and configuring an HMI tag

Follow these steps to configure the HMI tag for dynamization of the graphic property:

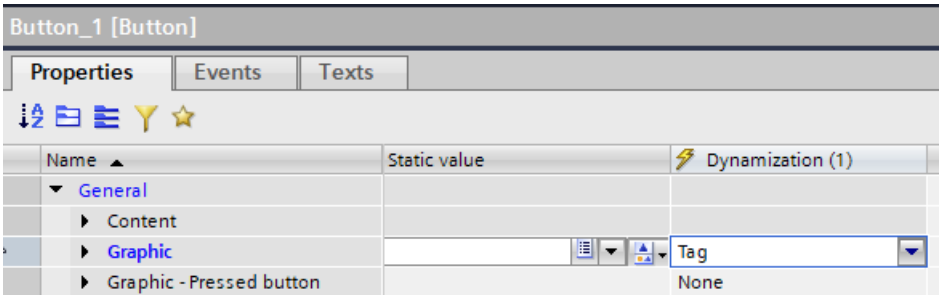
1. In the project tree under "Languages and resources > Project graphics", add a graphic, e.g. "Icon_filter", from your storage location.
2. Configure a "MyTag" tag of the data type "WString" in the project tree under "HMI tags".
3. In the Inspector window of the "MyTag" tag under "Properties > Properties > Values", enter the name of the added "Icon_filter" graphic in the "Start value" input field.



Dynamizing the "Graphic" property with the HMI tag

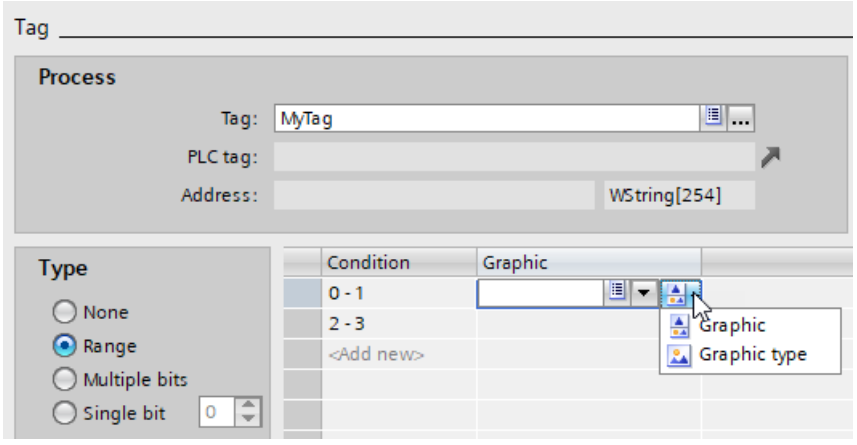
Follow these steps to dynamize the "Graphic" property at the "Button" object with the "MyTag" tag of the type "Area":

1. Click "Properties > Properties > General > Graphic" in the Inspector window of the screen editor. In the drop-down list of the "Dynamization" column, select the entry "Tag".

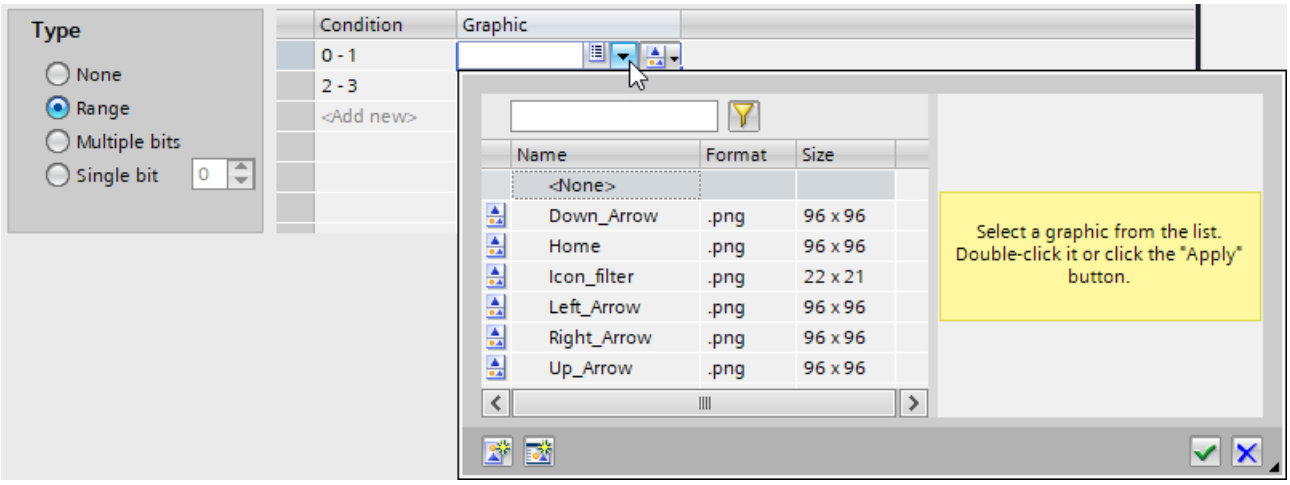


   The "Tag" dialog opens.

2. Click on the selection button ... under "Tag > Process > Tag".
   A dialog opens.
3. Add the "MyTag" tag.
4. Select the "Area" type under "Tag > Type". Define the conditions.
5. Select the graphic in the "Graphic" column.



6. Click on the selection button under "Tag > Type" in the "Graphic" column.
   A dialog opens.



   Select the graphic that you have stored in the project graphics.

7. Load the project into the device.

Result

All graphics in the project graphics are loaded into the runtime. You can use the name of the selected graphic in the tag. The graphic is displayed on the button in runtime.

If the "IntTag" tag is in the range "0 - 1" in runtime, the selected graphic is visible in the button.

Dynamizing a graphic property with a script

Introduction

You can dynamize a property containing a graphic with a script.

Task

In this example, you learn how to dynamize the "Graphic" property of the "Trend control" object with a script.

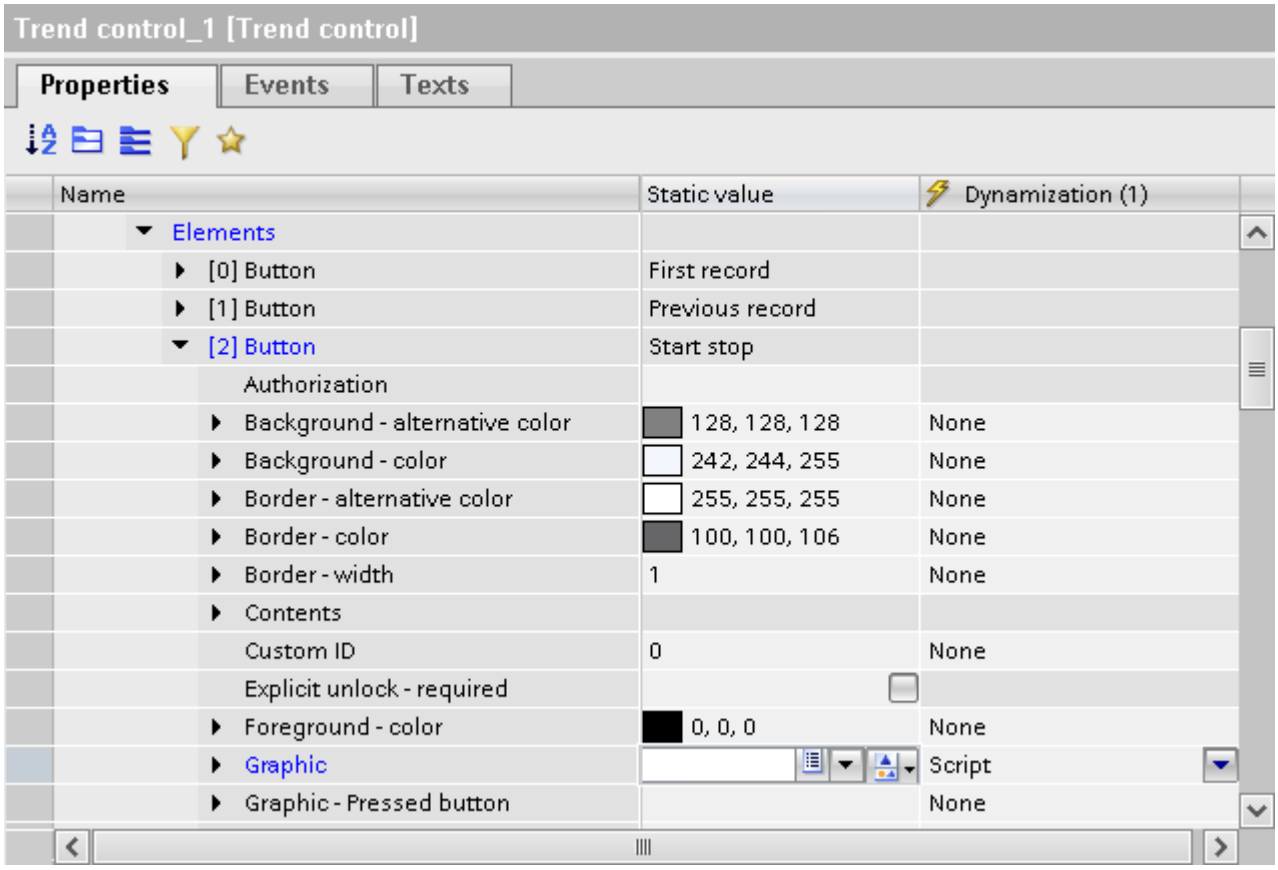
Requirement

- A project is open.
- A screen is configured.
- A trend control is configured in the screen.
- An HMI tag "MyTag" of the data type "Int" is configured with the start value "10".

Procedure

To dynamize the "Graphic" property of a trend control button with a script, follow these steps:

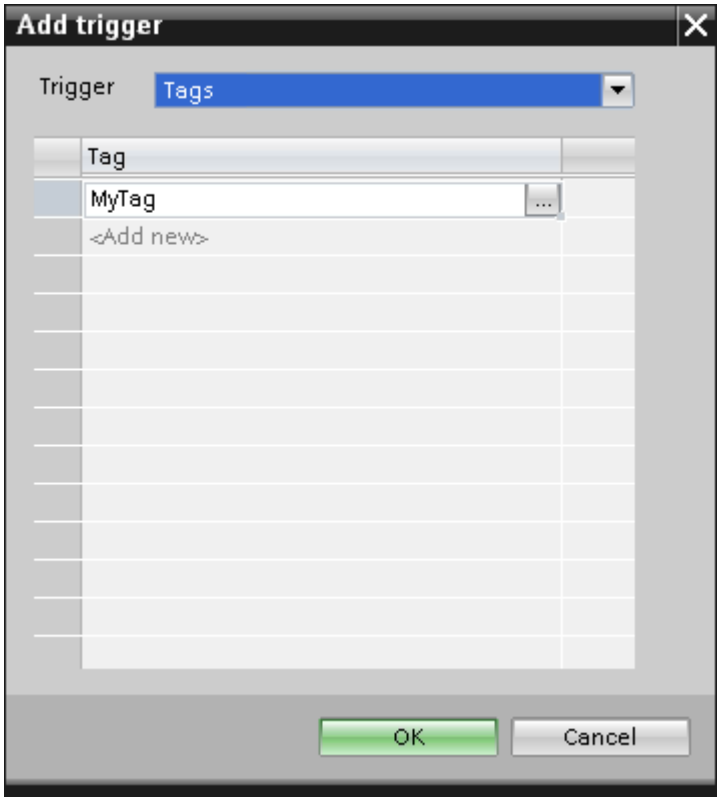
1. Select the trend control in the screen editor.
2. In the Inspector window, click on "Properties > Properties > Miscellaneous > Toolbar > Elements > [2] Button > Graphic". In the drop-down list of the "Dynamization" column, select the entry "Script".



The "Scripts" editor opens.

3. Click  .
A dialog opens.

4. Add the "MyTag" tag as a trigger of the script which initiates dynamization in runtime.



5. Add the following code in the "Scripts" editor:

Copy code

```
var value;
let tag = Tags("MyTag");
let tagValue = tag.Read();
HMIRuntime.Trace("Value of MyTag: " + tagValue);

switch (tagValue) {
case 10:
   value =
HMIRuntime.Resources.Graphics("GraphicCollection.Up_Arrow").Name;
   break;
case 20:
   value =
HMIRuntime.Resources.Graphics("GraphicCollection.Down_Arrow").Name;
   break;
}

return value;
```

Result

The graphic property is dynamized with a script. The return value of the script specifies the graphic of the trend diagram button in runtime.

If the "MyTag" HMI tag assumes the value "10" or "20" in runtime, the respective graphic is displayed in the button.

Dynamizing an object property with flashing using a script

Introduction

You can display objects as flashing in runtime. You can configure the flashing characteristics for each color setting of an object that supports flashing in the Inspector window and select the colors, the condition, and the flash rate.

Note

Flashing in runtime does not change the color value of the property.

Requirement

- A screen is open.
- An object is configured.
- An object property supports the dynamization type "Script".

Dynamizing an object property with flashing using a script

To dynamize an object property with flashing using a script, follow these steps:

1. Select the object.
2. In the Inspector window under "Properties", select the property for which you want to define the flashing characteristics, for example, "Background color".
3. In the "Dynamization" column, select the "Script" option. The right part of the Inspector window is displayed.
4. Enter a matching script.
   - To deactivate flashing, enter the following script:
     ```
     Screen.Items("Circle").PropertyFlashing("BackColor", false);
     ```
   - To activate flashing, enter the following script:
     ```
     Screen.Items("Circle").PropertyFlashing("BackColor", true);
     ```
   - To activate flashing and to define the properties, enter the following script:
     ```
     Screen.Items("Circle").PropertyFlashing("BackColor", true,
     HMIRuntime.Math.RGB(255, 0, 0), HMIRuntime.Math.RGB(0, 255, 0),
     UI.Enums.HmiFlashingRate.Fast);
     ```

Note

For dynamization of a faceplate, replace the term "Screen" with "Faceplate" in the script.

	**Efficiency tip**
You can find snippets for flashing in the shortcut menu of the "Scripts" editor under "Snippets > HMI Runtime > Screen"	

Result

The object property is dynamized with flashing with the dynamization type "Script". When the configured condition occurs in runtime, the object property flashes in the configured colors and at the set rate.

Dynamizing an object property with flashing with user-defined function

Introduction

You can display objects as flashing in runtime. You can configure the flashing characteristics for each color setting of an object that supports flashing in the Inspector window and select the colors, the condition, and the flash rate.

Note

Flashing in runtime does not change the color value of the property.

Requirement

- A screen is open.
- An object is configured.
- The object property supports the dynamization type "Flashing".

Dynamizing an object property with flashing with user-defined function

If you want to enable flashing via a user-defined function at an event, select the condition "Never" for the "Flashing" option at the relevant property under "Properties > Properties > Dynamization".

To dynamize an object property with flashing using a user-defined function, proceed as follows, for example:

1. Select the object that is to trigger flashing, e.g. a button.
2. In the Inspector window, under "Events", select the event to trigger flashing, e.g. "Press".
3. Select the "Convert function list to script" button.

4. To enable flashing, enter the following script, for example:
   `Screen.Items("flashObject").PropertyFlashing("BackColor", true);`
   - `flashObject` is an object that flashes when the user-defined function is triggered.
   - `BackColor` exemplifies the property that flashes.

   To activate flashing and to define the properties, enter the following script:
   - `Screen.Items("flashObject").PropertyFlashing("BackColor", true, HMIRuntime.Math.RGB(255, 0, 0), HMIRuntime.Math.RGB(0, 255, 0), UI.Enums.HmiFlashingRate.Fast);`

   "Flashing" can be applied to any property whose value is a color.

	**Efficiency tip**
You can find snippets for flashing in the shortcut menu of the "Scripts" editor under "Snippets > HMI Runtime > Screen"	

5. Select the object that you want to flash.
6. Select the property that is to flash, e.g. the background color.
7. Make the following setting for the property:
   - Condition: "Never"
   - If not already defined by the script, set different values for color and alternative color. Define the frequency.

The object property is dynamized with flashing by a user-defined function.

When the configured condition occurs in runtime, the object property flashes in the configured colors and at the set rate.

Note

For dynamization of a faceplate, replace the term "Screen" with "Faceplate" in the script.

Result

The object property is dynamized with flashing with a user-defined function. When the configured condition occurs in runtime, the object property flashes in the configured colors and at the set rate.

Dynamizing a screen property with a tag

You can dynamize the "Screen" property of a screen window with a tag. The tag can use the name or number of a screen.

Task

In this example, dynamize the "Screen" property of the "Screen window" object with a tag that uses a screen number.

You configure:

- Screen
- Screen window
- HMI tag "ScreenNumber"

Requirement

- A project is open.
- A screen is configured and set as the start screen.
- A screen window is configured in the start screen.

Configuring the HMI tag

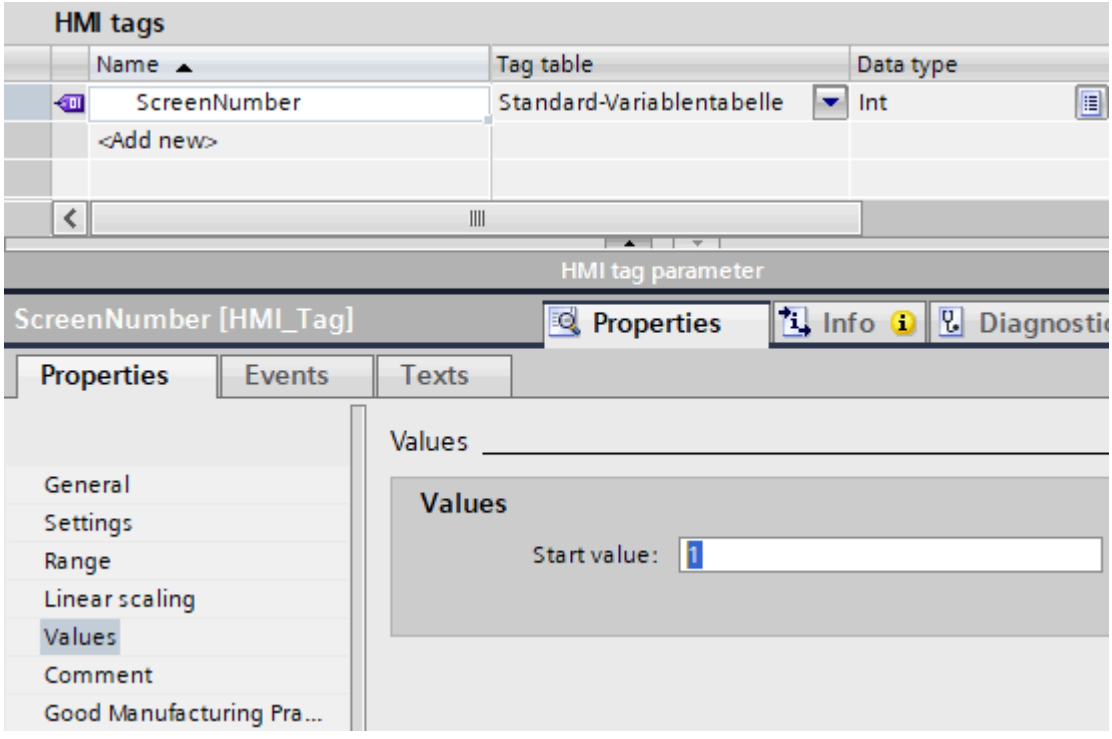
Follow these steps to configure the HMI tag for dynamization of the property:

1. Configure a "ScreenNumber" tag of the data type "Int" in the project tree under "HMI tags".

   Note

   The data type defines the type of screen referencing:
   - "Int": Dynamization uses the screen number of a screen.
   - "WString": Dynamization uses the name of a screen.

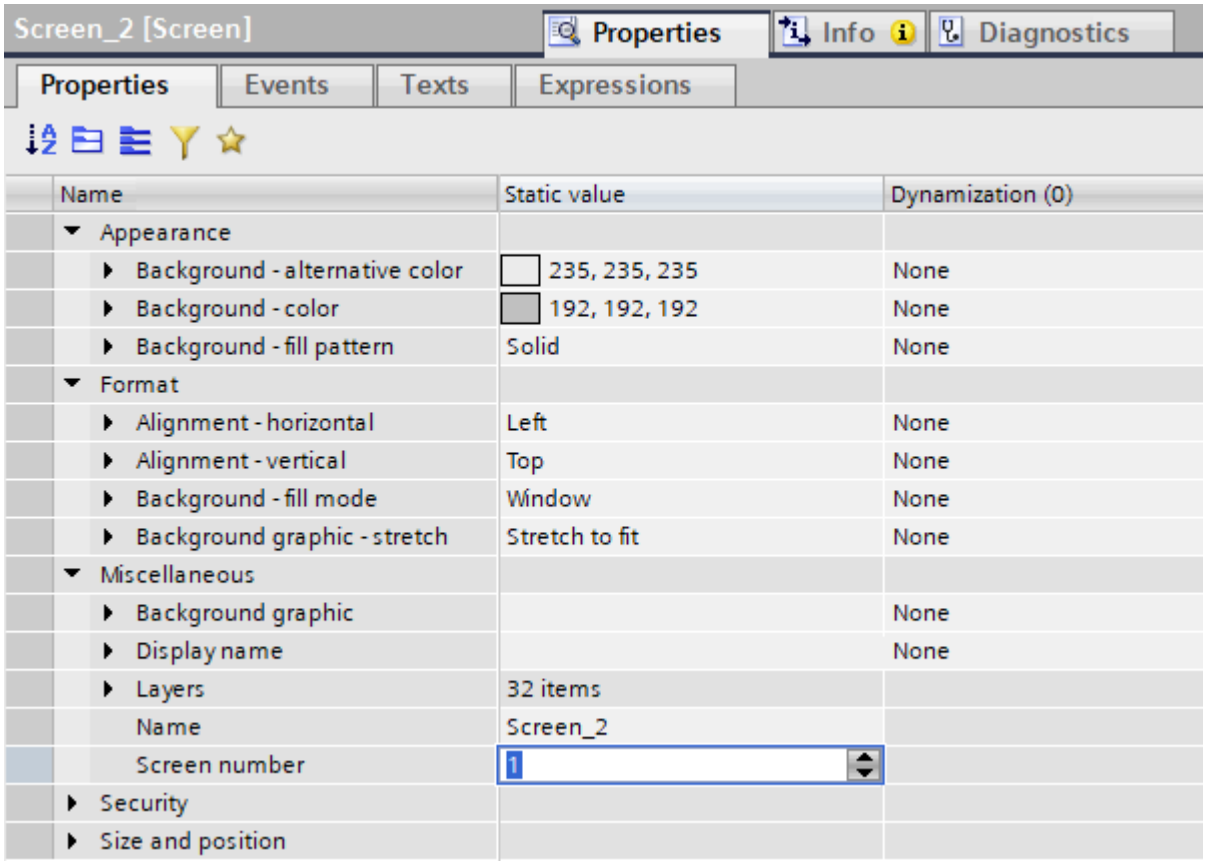
2. In the Inspector window of the "ScreenNumber" tag under "Properties > Properties > Values", enter the screen number "1" in the "Start value" input field.



Configuring a new screen

To configure a screen for display in the screen window, follow these steps:

1. Configure a new screen in the project navigation under "Screens".
2. In the Inspector window of the screen editor, enter the screen number "1" in "Properties > Properties > Miscellaneous > Screen number".



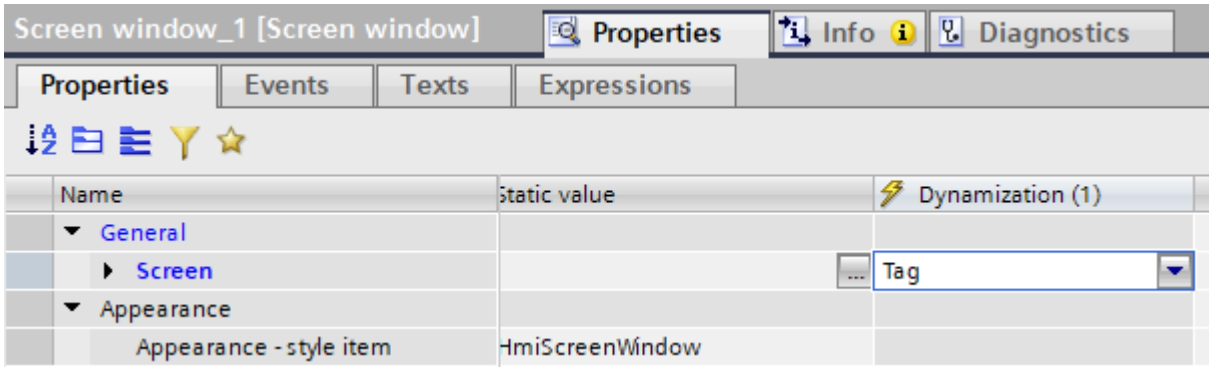
Note

The value of the screen number under all configured screens must be unique and greater than "0".

Dynamizing the "Screen" property with the HMI tag

Proceed as follows to dynamize the "Screen" property at the "Screen window" object with the tag "ScreenNumber":

1. In the screen editor of the start screen, select the object "Screen window".
2. In the Inspector window, click "Properties > Properties > General > Screen". In the drop-down list of the "Dynamization" column, select the entry "Tag".



The "Tag" dialog opens.

3. Click on the selection button ... under "Tag > Process > Tag".
   A dialog opens.
4. Add the "ScreenNumber" tag.
5. Download the project to the device.

Result

The screen with screen number "1" is displayed in runtime in the screen window.

If the integer value of the HMI tag "ScreenNumber" changes in runtime, the screen with the corresponding screen number is displayed in the screen window.

4.6 Trigger events

4.6.1 Basics on the events

Introduction

In runtime, an event triggers an action that you have configured on an object.

Basics on the events

You can configure one or multiple events on an object. You program a script on an event. The script defines an action that is executed in runtime when the operator triggers a specific event by operating the object.

The available events depend on the object being used.

Events

The following table shows the available events:

Event	Description
Activated	Mouse operation: Occurs when the operator presses the left or right mouse button. Touch operation: Occurs when the operator touches the screen with the finger. Keyboard operation: Occurs when the operator selects an object using the configured tab sequence. The "Activated" event is only used to detect whether an object was selected. The event does not trigger a password prompt. For this reason, do not use the "Activated" event if you want to configure access protection on the function call of the object.
Deactivated	Occurs when the operator takes the focus off an object by activating a different object. System functions or user-defined functions on the "Deactivated" event of an object are not executed with a screen change. The "Deactivated" event is only used to detect whether an object was deselected. The event does not trigger a password prompt. For this reason, do not use the "Deactivated" event if you want to configure access protection on the function call of the object.
Press	Mouse operation: Occurs after the "Activated" event when the operator presses the left or right mouse button. Touch operation: Occurs when the operator touches the screen with the finger. Keyboard operation: Occurs when an object has the focus, and the operator presses either the <Return> or <Space> key.
Release	Mouse operation: Occurs when the operator releases the left or right mouse button. Touch operation: Occurs when the operator takes the finger off the screen. When a motion is detected, the "Release" event is triggered at the limit of a different screen object. Keyboard operation: Occurs when the operator releases one of the buttons <Return> or <Space>.
Press key	Keyboard operation: Occurs when the operator presses a key on the keyboard. The event is not triggered if one of the buttons <Return> or <Space> is pressed.
Release key	Keyboard operation: Occurs when the operator releases a key on the keyboard.
Trigger hotkey	Occurs when the operator presses a hotkey on the keyboard.
Click left mouse button	Mouse operation: Occurs when the system detects a click with the left mouse button, i.e. shortly after the "Release" event. Touch operation: Occurs after the "Release" event has occurred and if less than a second has elapsed since the "Press" event.
Click right mouse button	Mouse operation: Occurs when the system detects a click with the right mouse button, i.e. immediately after the "Release" event. Touch operation: Occurs after the "Release" event has occurred and if more than a second has elapsed since the "Press" event.
Loaded	Occurs when a screen is fully loaded in runtime after a screen change.
Cleared	Occurs when the active screen is not loaded in runtime.
Connected	Occurs when the object has been successfully initialized and the data connection to the controller has been established.

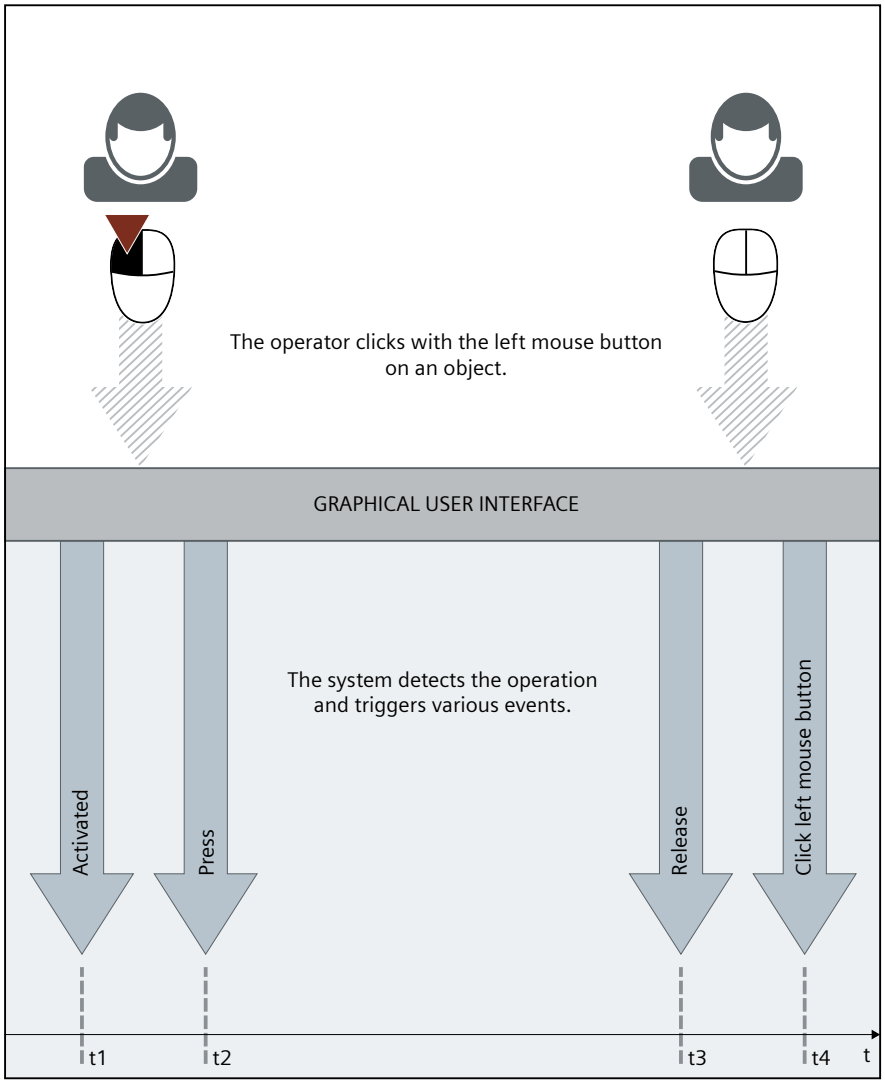
Event	Description
Status changed	Occurs when the state of the switch changes, for example, from "On" to "Off".
Command fired	Occurs when the operator has actuated a button in the toolbar or information bar.
Gesture detected	Occurs when the operator performs a touch gesture.
Play	Occurs when the video or audio file is being replayed.
Pause	Occurs when playing of the video or audio file is paused.
Playback finished	Occurs when the video or audio file has been played.
Selection changed	Occurs when the selection is changed.
Expand	Occurs when all plant objects under a node are displayed.
Expand all	Occurs when all plant objects are displayed.
Minimize	Occurs when all plants objects under a node are hidden.
Minimize all	Occurs when all plant objects are hidden.

Example

The figure below shows the event "Click left mouse button" as an example:

When you want to trigger an event by clicking with the left mouse button, follow these steps:

1. Click on the object with the left mouse button.
   The system detects the operation and triggers the "Activated" and "Pressed" events.
2. Release the left mouse button.
   The system detects the operation and triggers the events "Release" and "Click left mouse button".



4.6.2 Triggering "Activated" and "Deactivated" events

Introduction

You can trigger the event "Activated" and "Deactivated" through movement with the left or right mouse button, on the touchscreen or through keyboard operation.

When an object has the focus, the "Activated" event is automatically triggered before the "Pressed" event.

"Activated" event

You have the following options to trigger the "Activated" event:

- Mouse operation: Press the left or right mouse button.
- Touch operation: Touch the screen using your finger.
- Keyboard operation: Select an object using the configured tab sequence.

"Deactivated" event

You trigger the "Deactivated" event when you take the focus off an object by activating a different object.

In case of a screen change, the "Deactivated" event of an object is triggered but the system functions or user-defined functions on the "Deactivated" event are not executed.

The "Deactivated" event is only used to detect whether an object was deselected. The event does not trigger a password prompt. For this reason, do not use the "Deactivated" event if you want to configure access protection on the function call of the object.

4.6.3 Triggering a "Press" event

Introduction

You can trigger the "Press" event through movement with the left or right mouse button, on the touchscreen or through keyboard operation.

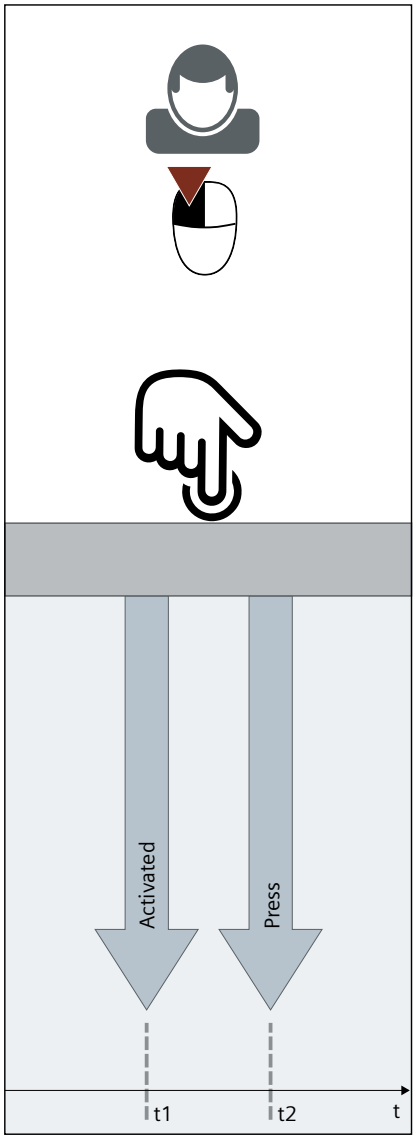
When an object has the focus, the "Activated" event is automatically triggered before the "Press" event.

"Press" event

You have the following options to trigger the "Press" event:

- Mouse operation: Occurs when you press the left or right mouse button.
- Touch operation: Occurs when you touch the screen with your finger.
- Keyboard operation: Occurs when an object has the focus, and you press either the <Return> or <Space> key.

The figure below shows the "Activated" and "Press" events:



4.6.4 Triggering a "Release" event

Introduction

You can trigger the event "Release" through movement with the left mouse button or on the touchscreen.

"Release" event by moving in the object with the mouse

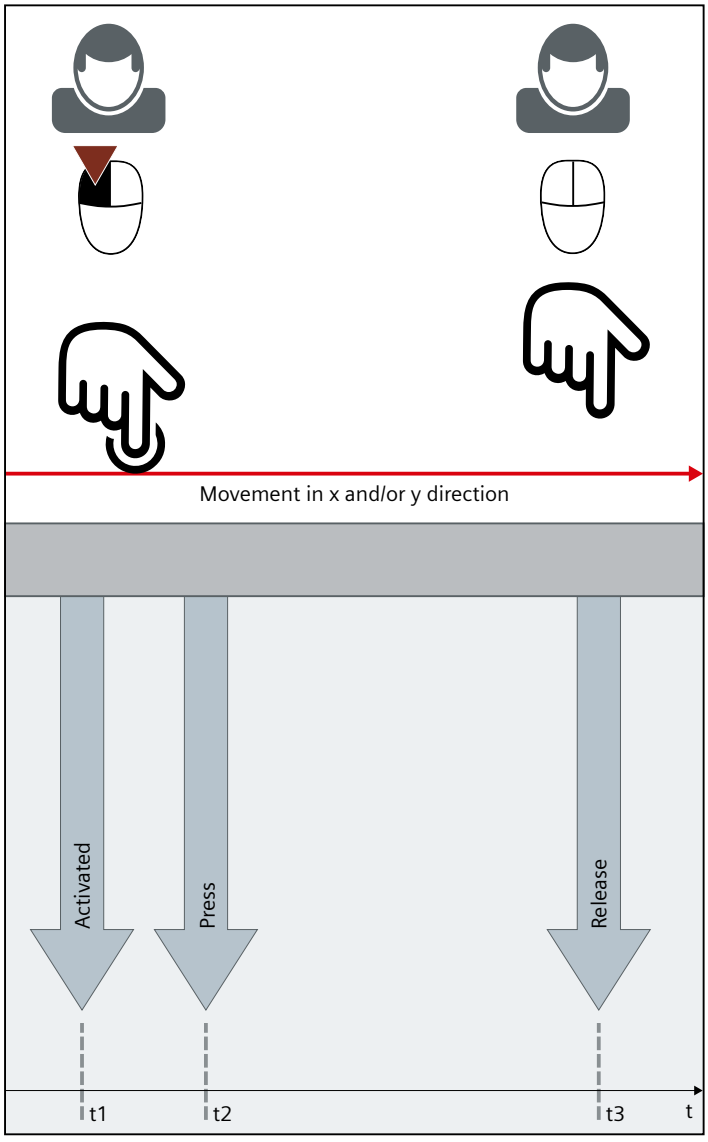
When you want to trigger the event "Release" through the mouse operation, follow these steps:

1. Click on the object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Move with the mouse in the object in x and/or y direction.
3. Release the left mouse button.
   Releasing triggers the "Release" event.
   The "Click left mouse button" event is not triggered.

"Release" event by movement in the object using touch control

When you want to trigger the event "Release" through the touch operation, follow these steps:

1. Touch the object using your finger.
   Touching triggers the "Activated" and "Press" events.
2. Move with your finger in the object in x and/or y direction.
3. Release the finger.
   The releasing triggers the "Release" event.



"Release" event by moving over the object boundary with the mouse

When you want to trigger the event "Release" through the mouse operation, follow these steps:

1. Click on the object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Move with the mouse in x and/or y direction beyond the object limit.
3. Release the left mouse button.
   Releasing triggers the "Release" event.
   The "Click left mouse button" event is not triggered.

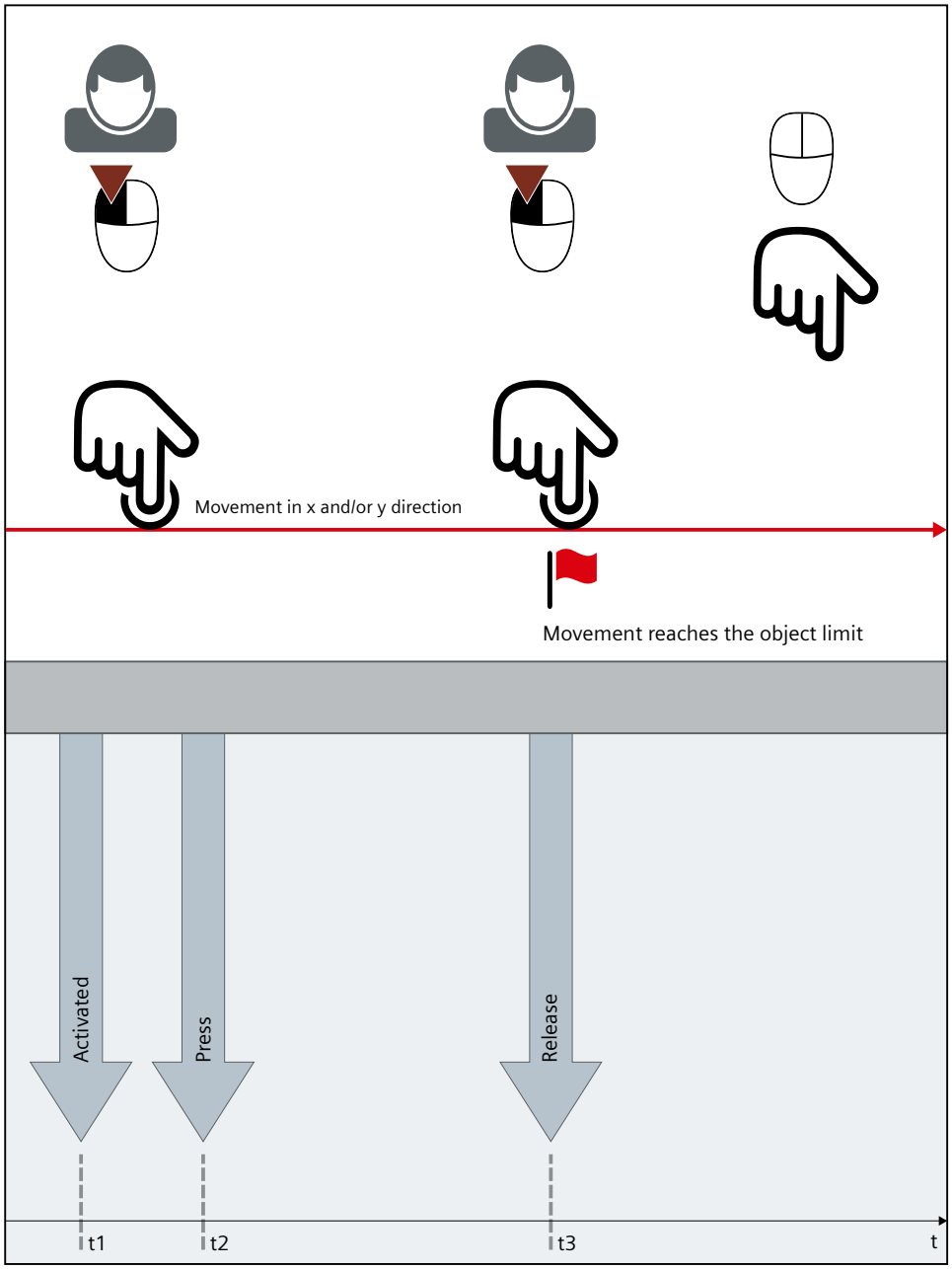
If you continue moving with the finger in the object instead of releasing it, no additional event is triggered.

After you release the finger or the mouse button outside of the object, no additional event is triggered.

"Release" event by movement over the object boundary using touch operation

When you want to trigger the event "Release" through the touch operation, follow these steps:

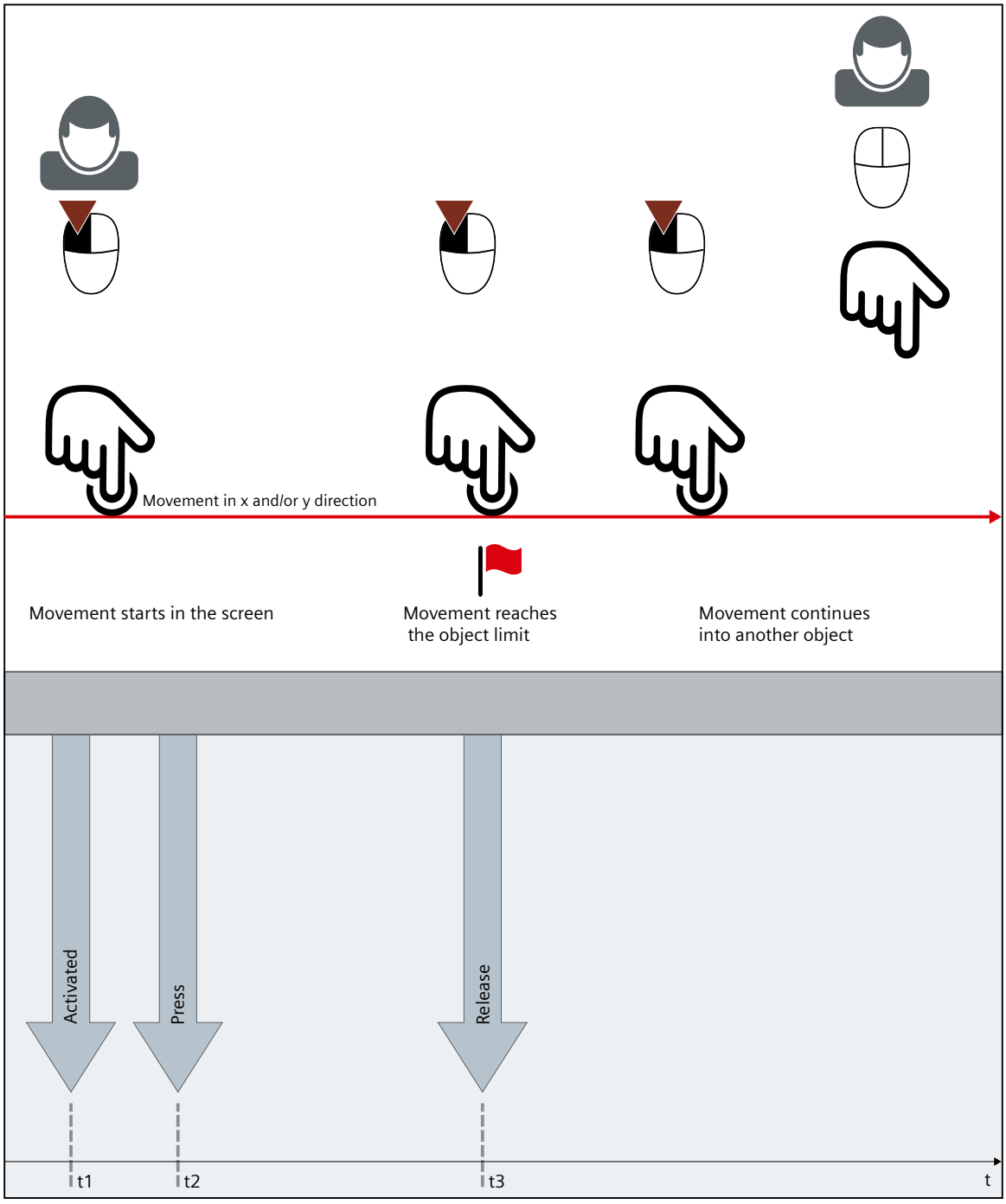
1. Touch the object using your finger.
   Touching triggers the "Activated" and "Press" events.
2. Move with your finger in the object in x and/or y direction beyond the object limit.
3. Release the finger.
   The releasing triggers the "Release" event.



"Release" event by movement into another object

Releasing the left mouse button or finger after reaching the object boundary triggers the "Release" event.

If you move with the left mouse button or finger in another object, no further event will be triggered.



Objects with a slide bar

When you click on objects with a slide bar, for example, slider or bar, with the left mouse button or touch them on the touchscreen, the movement of the slide bar is started. The "Release" event is not triggered at the object limit but only after the release.

The "Click left mouse button" event is not triggered.

4.6.5 "Press key" and "Release key" events:

Introduction

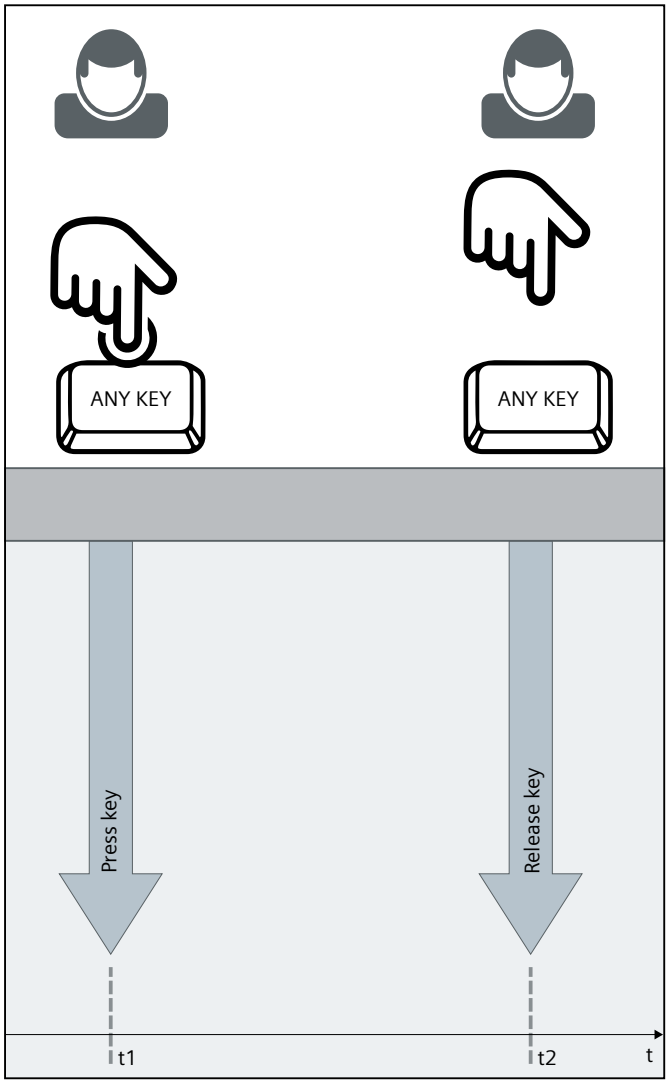
You can trigger the "Press key" and "Release key" events by keyboard action.

The behavior of the <Return> and <Space> keys differs from the behavior of the other keys on the keyboard.

Using key on the keyboard

If you want to trigger the "Release key" event by keyboard action, follow these steps:

1. Press a key on the keyboard.
   The "Press key" event is triggered.
2. Release the key.
   The "Release key" event is triggered.

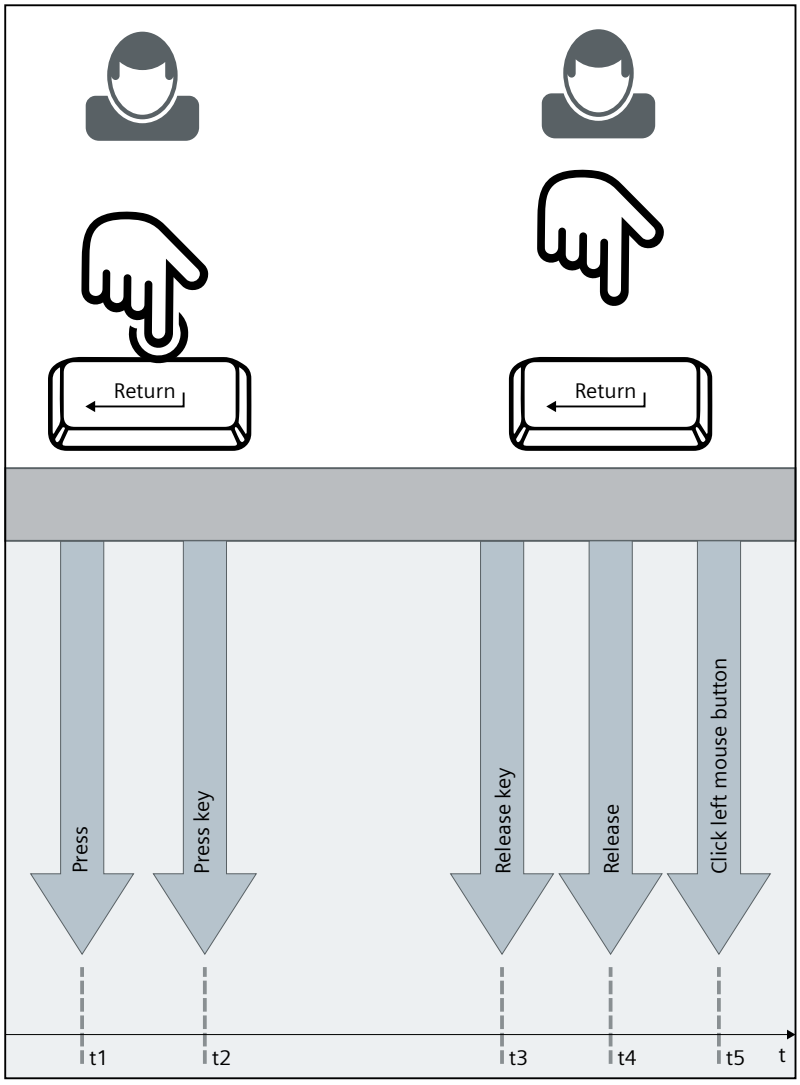


Using the <Return> or <Space> key

The keys <Return> and <Space> can trigger direct actions, for example, for button or rectangle.

When you want to trigger the events through the keys <Return> or <Space>, follow these steps:

1. Select an object on which you have configured an event.
2. Press the <Return> or <Space> key.
   The "Press" and "Press key" events are triggered.
3. Release the key.
   The "Release key", "Release" and "Click left mouse button" events are triggered.



Objects with selection elements

In the objects with selection elements, e.g. check box, the <Return> key behaves like any key.

You can use the <Space> key, for example to change the selection from "On" to "Off" or to trigger an event.

4.6.6 Trigger "Click left mouse button" event

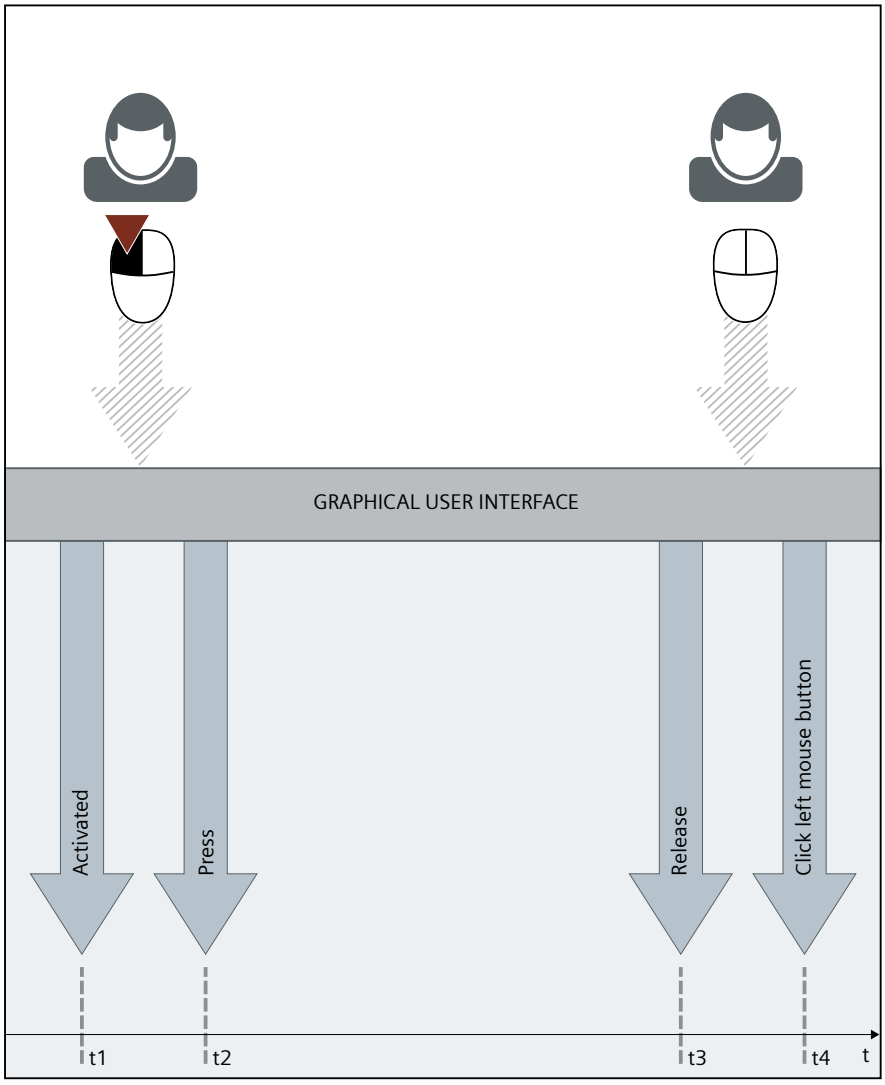
Introduction

You can trigger the event "Click left mouse button" by clicking with the left mouse button.

"Click left mouse button" event

When you want to trigger the "Click left mouse button" event through mouse action, follow these steps:

1. Click on the object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release" and "Click left mouse button" events.



4.6.7 Trigger "Click right mouse button" event

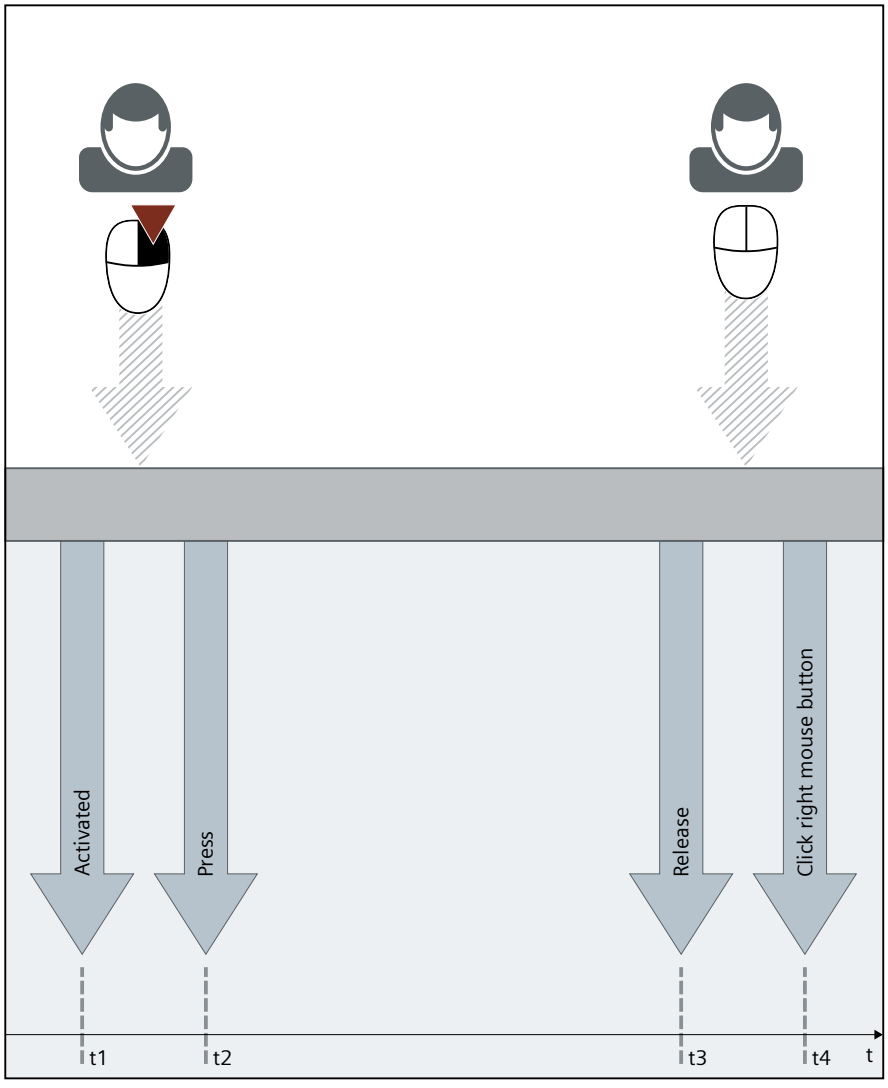
Introduction

You can trigger the event "Click right mouse button" by clicking with the right mouse button.

"Click right mouse button" event

When you want to trigger the event "Click right mouse button" through mouse operation, follow these steps:

1. Click on the object with the right mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the right mouse button.
   Releasing triggers the "Release" and "Click right mouse button" events.



4.6.8 "Loaded" event

The "Loaded" event occurs when a screen is fully loaded after a screen change or runtime start.

4.6.9 "Cleared" event

The "Cleared" event occurs when the active screen is not loaded in runtime.

4.6.10 "Connected" event

The "Connected" event occurs when the data connection between an object, for example a trend control, and the controller has been established.

4.6.11 Triggering the "Status changed" event

Introduction

You can trigger the "Status changed" event by operating the "Switch" object.

Trigger "Status changed" event by mouse operation

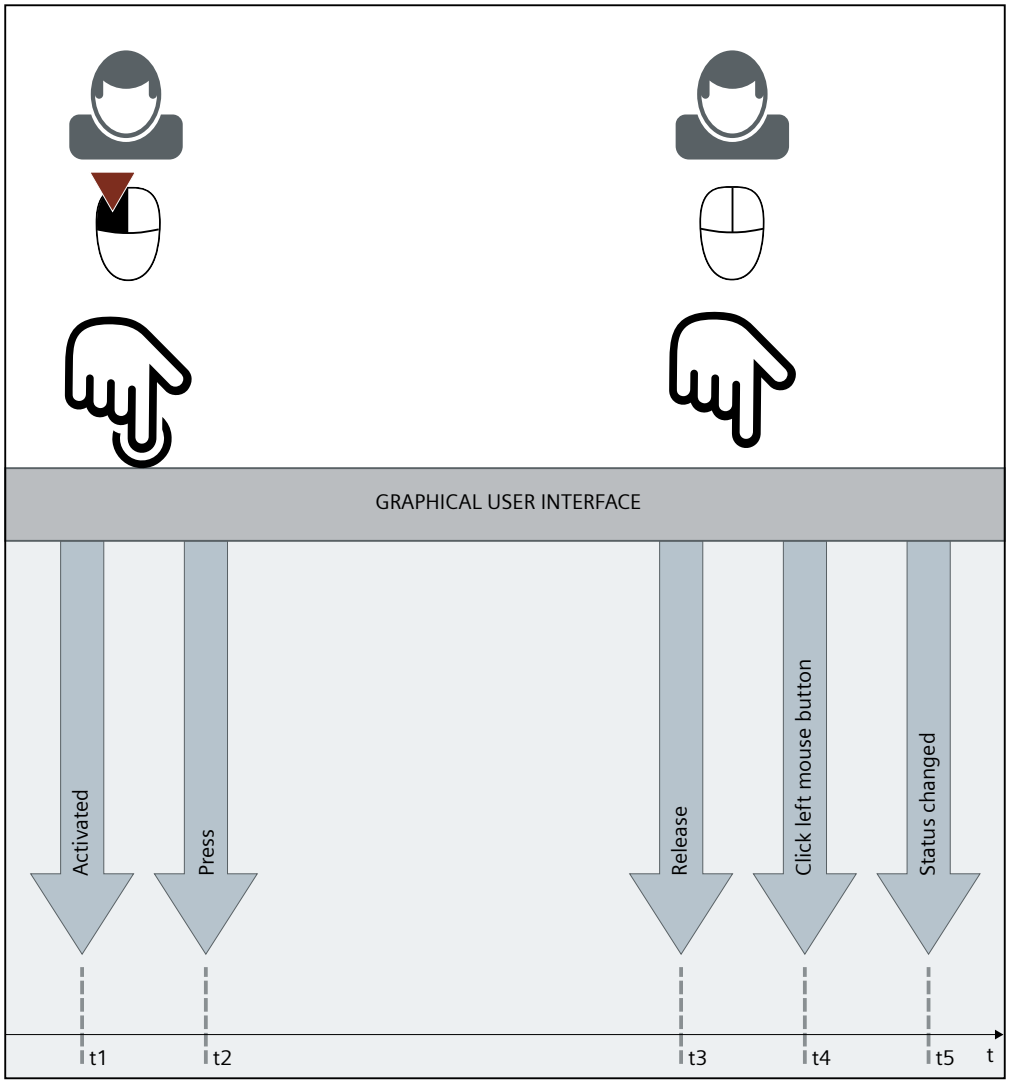
When you want to trigger the "Status changed" event through mouse operation, follow these steps:

1. Click with the left mouse button on the "Switch" object.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and "Status changed" events.

Trigger "Status changed" event by touch operation

When you want to trigger the "Status changed" event using touch operation, follow these steps:

1. Touch the object using your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger.
   Releasing triggers the "Release", "Click left mouse button" and "Status changed" events.



4.6.12 Trigger "Command fired" event

Introduction

You can trigger the "Command fired" event by operating a button in the object, e.g. "Alarm control".

Trigger "Command fired" event by mouse operation

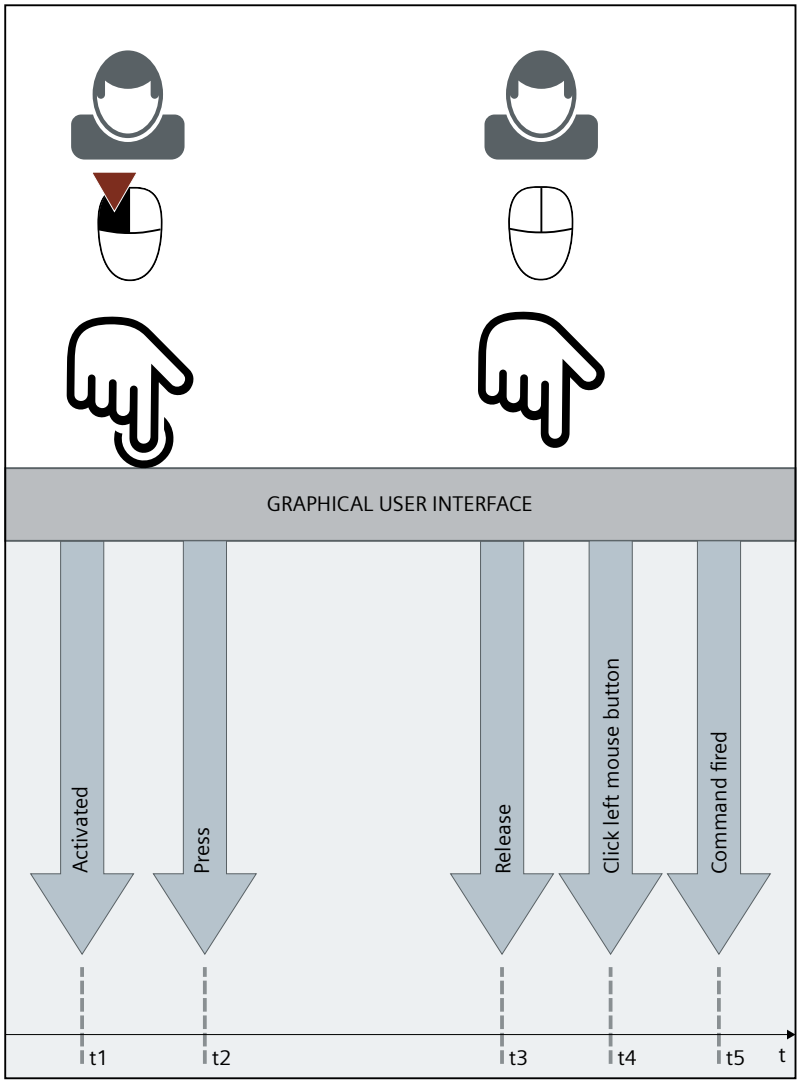
When you want to trigger the event "Command fired" through mouse operation, follow these steps:

1. Click on a button in the "Alarm control" object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and "Command fired" events.

Triggering a "Command fired" event by touch operation

When you want to trigger the "Command fired" event through touch operation, follow these steps:

1. Touch a button in the "Alarm control" object with your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger.
   Releasing triggers the "Release", "Click left mouse button" and "Command fired" events.



4.6.13 Trigger "Gesture detected" event

Introduction

The "Gesture detected" event occurs when the operator has performed a touch gesture.

Distinguish gestures

The "Touch Area" object distinguishes between the following gestures:

- Right
- Left

- Up
- Down

Programming a J-script

To distinguish between the gestures, program a J-Script that evaluates the gesture.

1. Click  in the Inspector window under "Properties > Events > Gesture detected".
2. Copy the code example into the programming window.

Code example

```
export function Touch_area_1_OnGestureDetected(item, gesture) {
// value of tag ‚MyTag1‘ will be set depending on the detected gesture
if(gesture == UI.Enums.HmiGesture.SwipeRight)
{
UI.RootWindow.Screen = 'ScreenRight';
let tag1 = Tags('tag1');
tag1.Write(1); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.SwipeLeft)
{
UI.RootWindow.Screen = 'ScreenLeft';
let tag1 = Tags('tag1');
tag1.Write(2); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.SwipeUp)
{
UI.RootWindow.Screen = 'ScreenUp';
let tag1 = Tags('tag1');
tag1.Write(3); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.SwipeDown)
{
UI.RootWindow.Screen = 'ScreenDown';
let tag1 = Tags('tag1');
tag1.Write(4); //write value '1234' to tag 'MyTag1'
}
if(gesture == UI.Enums.HmiGesture.Unknown)
{
let tag1 = Tags('tag1');
tag1.Write(0); //write value '1234' to tag 'MyTag1'
}
}
```

4.6.14 Triggering events through touch operation

Introduction

You can trigger events by touching the object on the touchscreen.

Note

Touch gestures and "Press" and "Release" events are not supported for client access of a PC with a touchscreen to a SmartServer.

Limit value

The time between touching and releasing the finger on the touchscreen decides which of the events is triggered.

If the time between touching and releasing (t3 – t2) the object is less than the specified limit value, releasing the pressed finger from the touchscreen triggers the event "Click left mouse button".

If the time between touching and releasing (t3 – t2) the object is greater than the specified limit value, releasing the pressed finger from the touchscreen triggers the event "Click right mouse button".

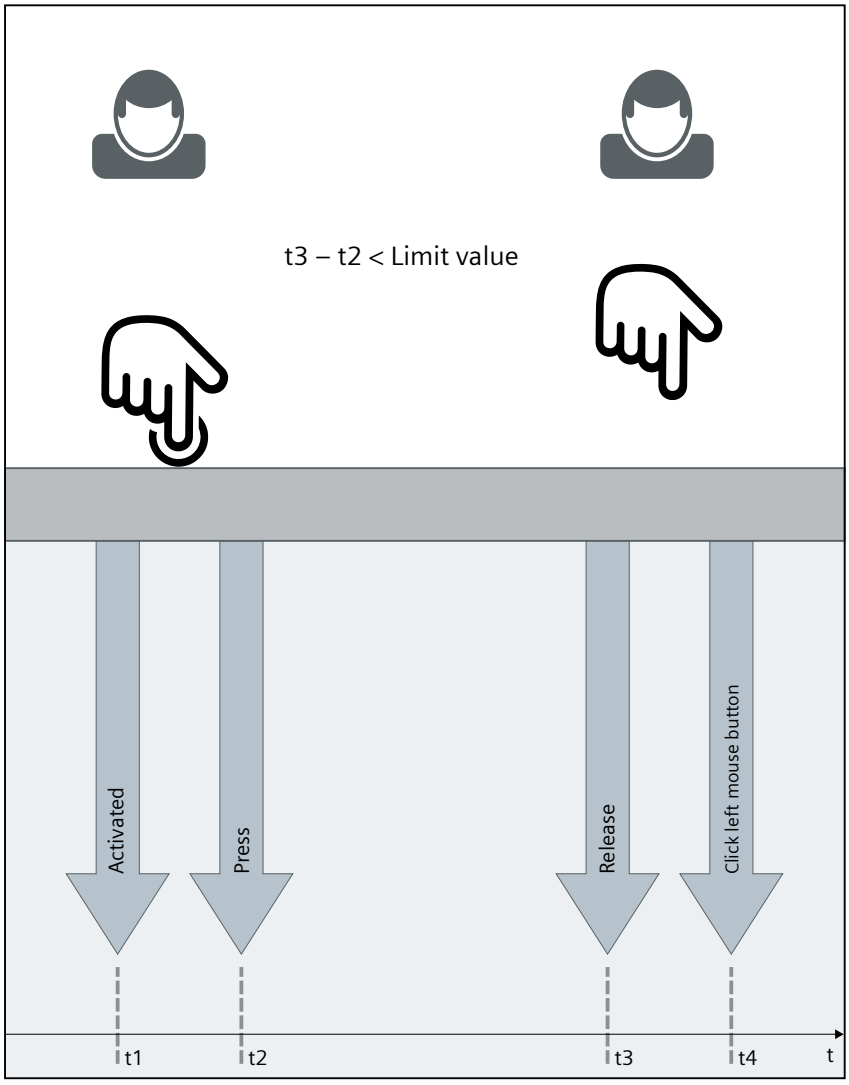
Note

If not specified by the operating system or the Web browser, the limit value is 1 000 ms.

Simulating clicking with the left mouse button

If you want to trigger the "Click left mouse button" event through touch operation, follow these steps:

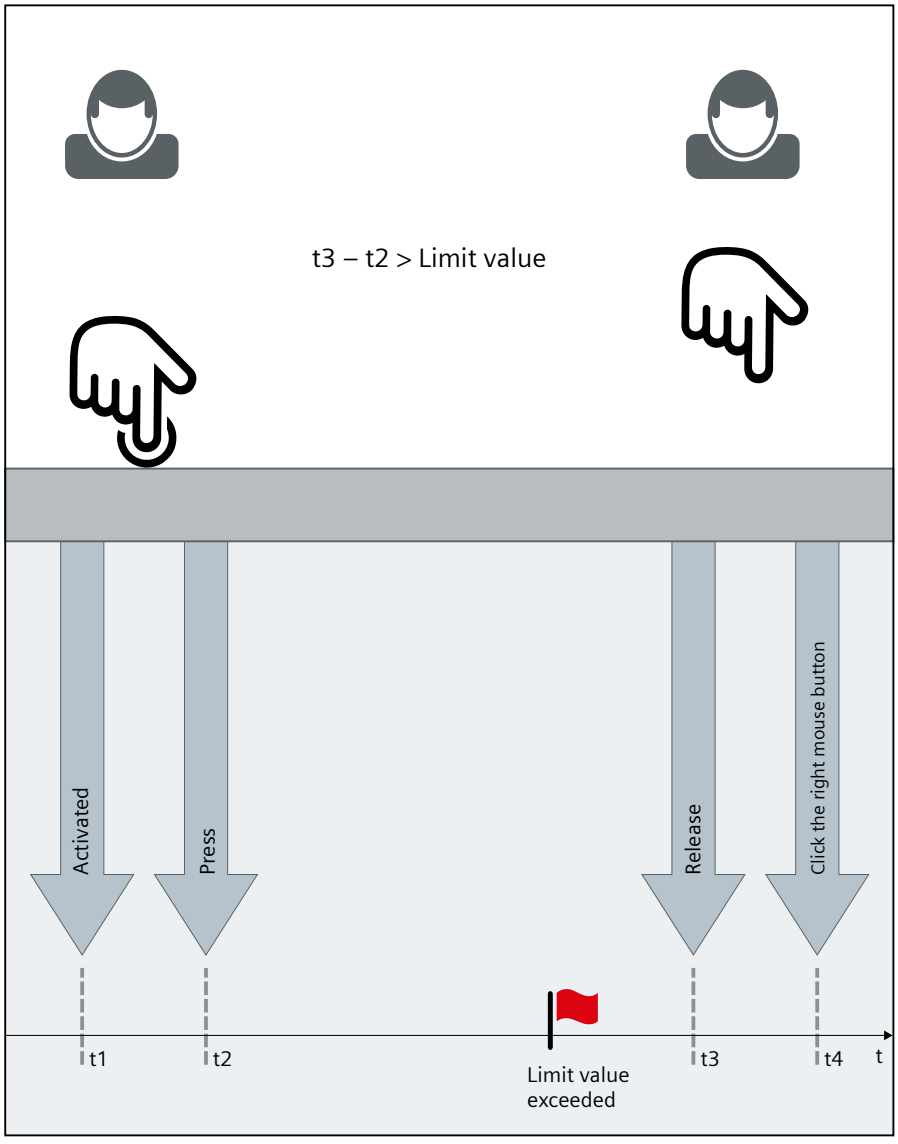
1. Touch the object using your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger within the limit value.
   Releasing triggers the "Release" and "Click left mouse button" events.



Simulating clicking with the right mouse button

If you want to trigger the "Click right mouse button" event through touch operation, follow these steps:

1. Touch the object using your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger after the limit value has expired.
   Releasing triggers the "Release" and "Click right mouse button" events.



4.6.15 Example: Configure the system function "Screen change"

Task

In this example you configure the system function "Screen change".

Requirement

- A project is open.
- A screen is configured.

Configure the system function "Screen change"

To configure a "Screen change" system function, follow these steps:

1. Create a screen "Screen1" and drag, for example, a button into the screen.
2. Create a screen "Screen2" and drag, for example, a circle into the screen.

Configuring "Screen1"

To add the "Click left mouse button" event to the button in "Screen1", follow these steps:

1. Select the button and click "Click left mouse button" under "Events".
2. Click "Add script". A "Script" window is opened.
3. Click on the first curly bracket in the script. The bracket turns green.
4. Under "Code templates > HMIRuntime > Screen", select the function "Change base screen".
5. Drag the function to the bracket. Enter "Screen2" in the script.
6. Click "OK".

Configuring "Screen2"

To add the "Click left mouse button" to the circle in "Screen2", follow these steps:

1. Select the circle and click "Click left mouse button" under "Events".
2. Click "Add script". A "Script" window is opened.
3. Click on the first curly bracket in the script. The bracket turns green.
4. Under "Code templates > HMIRuntime > Screen", select the function "Change base screen".
5. Drag and drop the function to the bracket, and write "Screen1" in the script.
6. Click "OK".

Loading the project and starting runtime

To load the project and start runtime, follow these steps:

1. Load the project under "Project > Device".
2. Start runtime.
3. Click on the button with the left mouse button. You switch to "Screen2".

Result

When you click on the circle with the left mouse button, you switch to "Screen1". When you click on the button with the left mouse button, you switch to "Screen2".

The system function "Screen change" has been configured.

4.6.16 Events on the "Media Player" object

4.6.16.1 Trigger "Pause" event

Introduction

You can trigger the "Pause" event by operating the "Pause button" in the "Media Player" object.

Triggering a "Pause" event by mouse operation

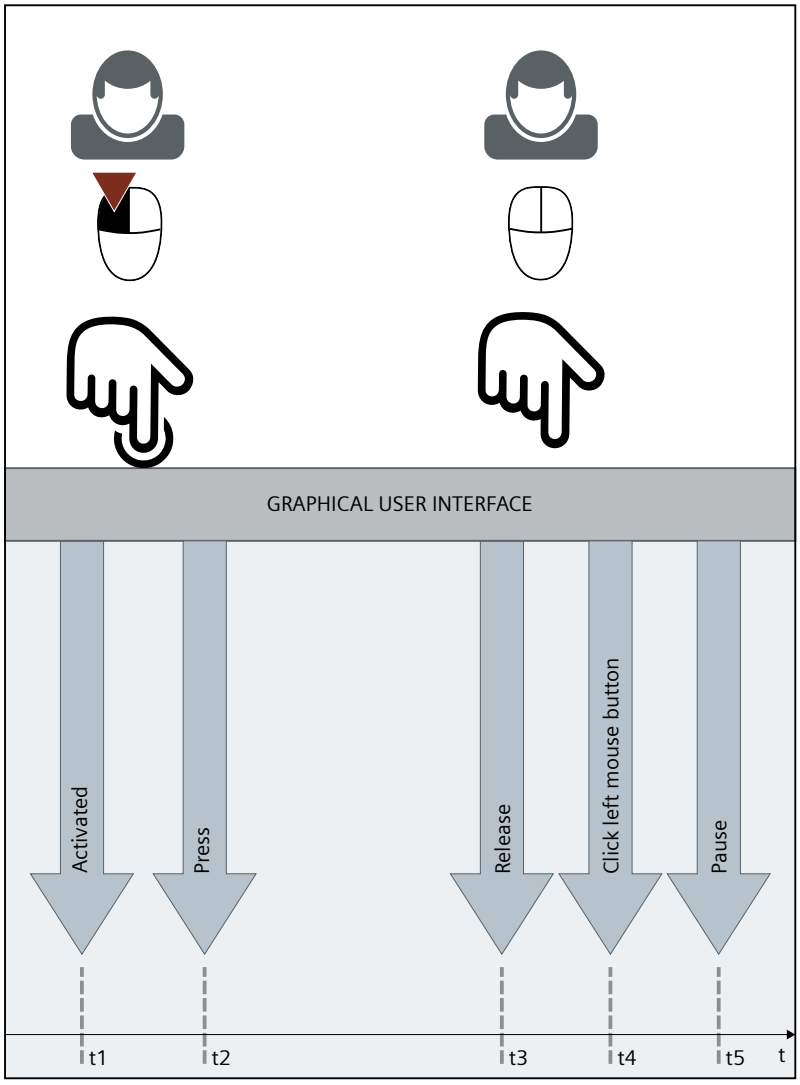
If you want to trigger the "Pause" event through mouse operation, follow these steps:

1. Click on the "Pause" button in the "Media Player" object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and "Pause" events.

Triggering a "Pause" event by touch operation

If you want to trigger the "Pause" event through the touch operation, follow these steps:

1. In the "Media Player" object, touch the "Pause" button with your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger.
   Releasing triggers the "Release", "Click left mouse button" and "Pause" events.



4.6.16.2 Triggering a "Play" event

Introduction

You can trigger the "Play" event by operating the "Play" button in the "Media Player" object.

Trigger "Play" event by mouse operation

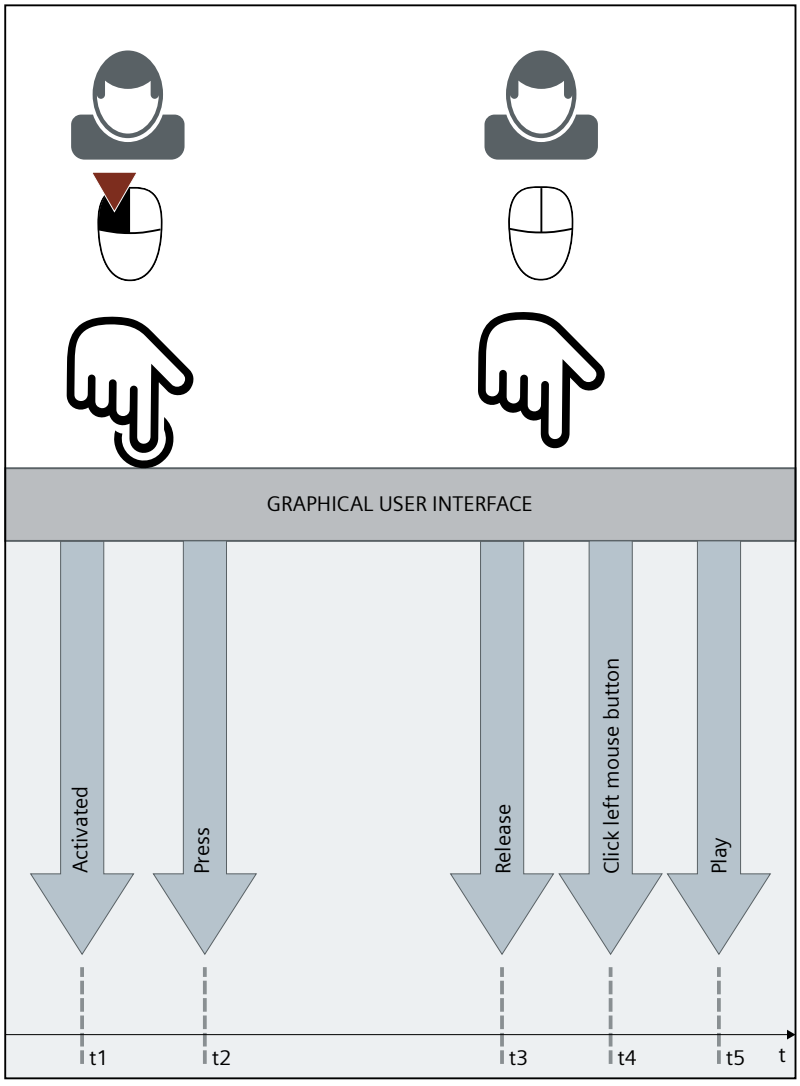
If you want to trigger the "Play" event by mouse operation, follow these steps:

1. Click on the "Play" button in the "Media Player" object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and "Play" events.

Triggering a "Play" event by touch operation

If you want to trigger the "Play" event through the touch operation, follow these steps:

1. In the "Media Player" object, touch the "Play" button with your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger.
   Releasing triggers the "Release", "Click left mouse button" and "Play" events.



4.6.16.3 Triggering a "Playback finished" event

Introduction

You can trigger the "Playback finished" event by operating the "Stop" button in the "Media Player" object.

Triggering a "Playback finished" event by mouse operation

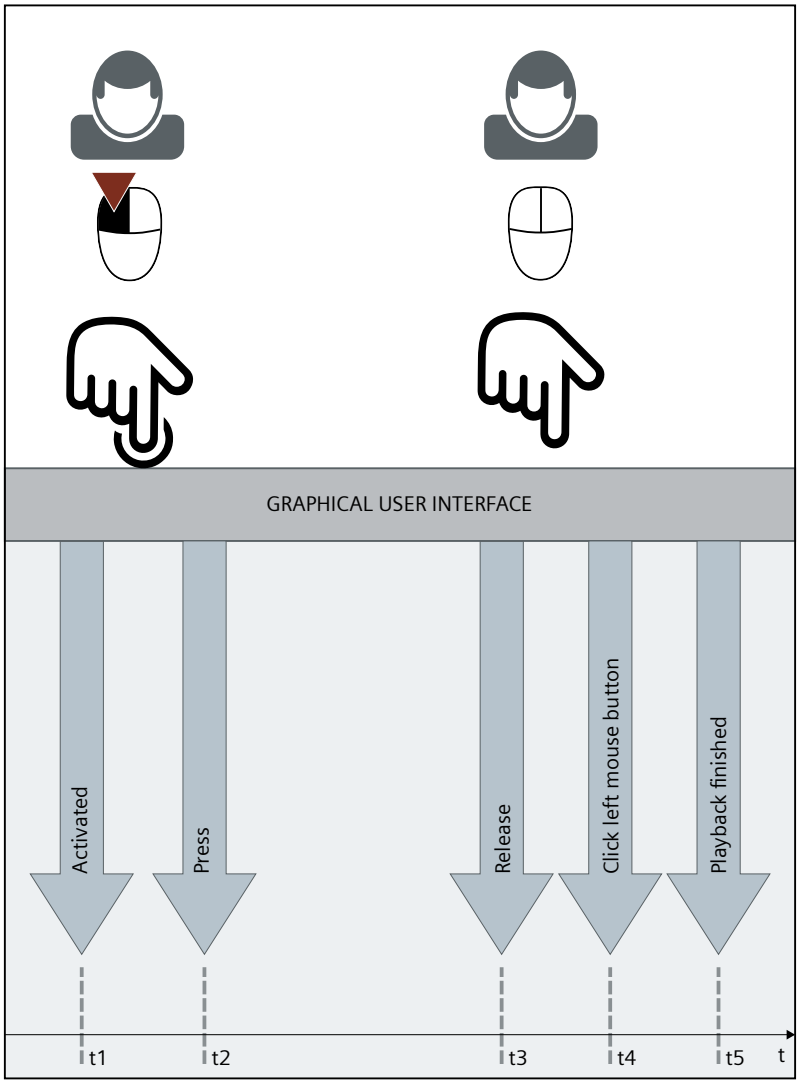
If you want to trigger the "Playback finished" event by mouse operation, follow these steps:

1. Click on the "Stop" button in the "Media Player" object with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and "Playback finished" events.

Triggering a "Playback finished" event by touch operation

If you want to trigger the event "Playback finished" by the touch operation, follow these steps:

1. In the "Media Player" object, touch the "Stop" button with your finger.
   Touching triggers the "Activated" and "Press" events.
2. Release the finger.
   Releasing triggers the "Release", "Click left mouse button" and "Playback finished" events.



4.6.17 Events at the "Plant overview" object

4.6.17.1 Triggering a "Selection changed" event

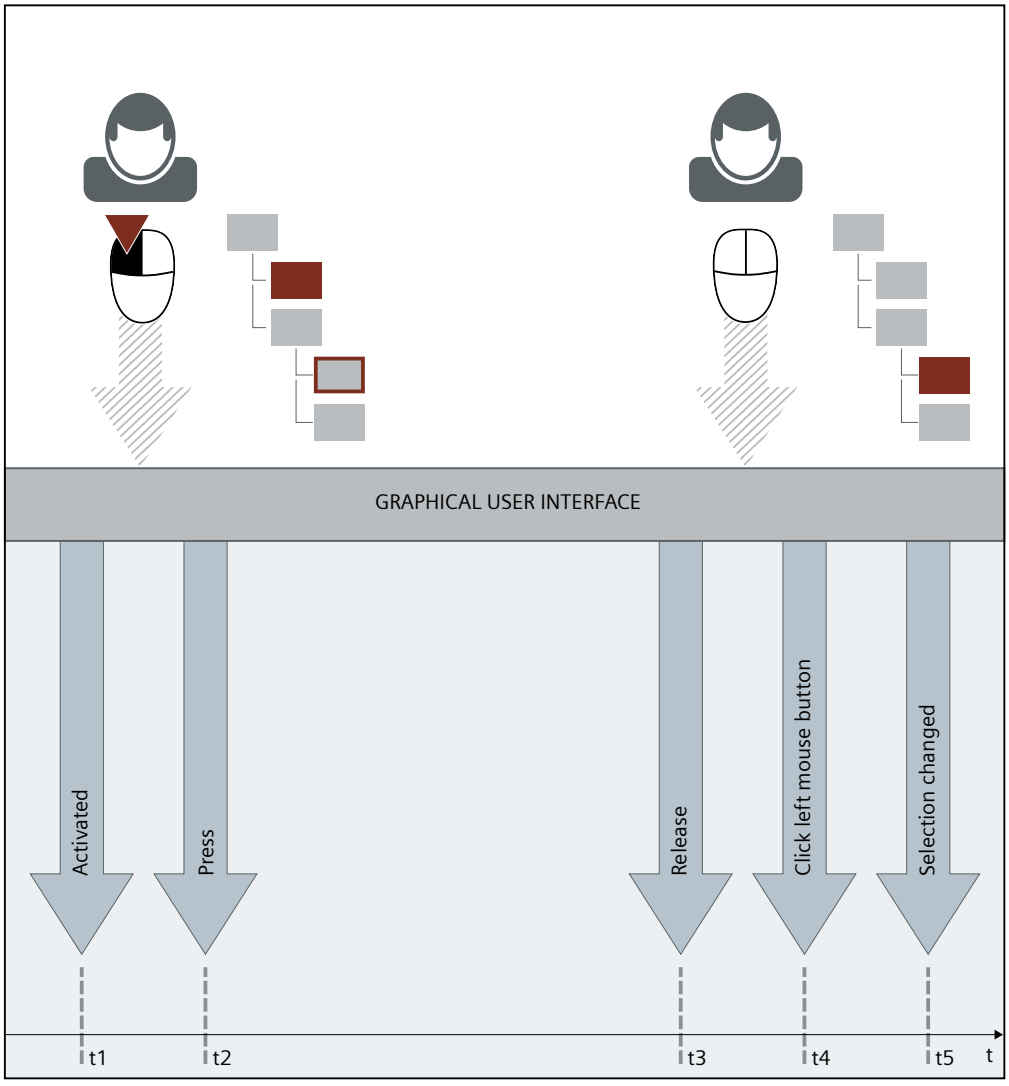
Introduction

You can trigger the "Selection changed" event in the "Plant overview" object by clicking the left mouse button.

Triggering a "Selection changed" event

If you want to trigger the "Selection changed" event through mouse operation, follow these steps:

1. Click on a node with the left mouse button.
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and finally "Selection changed" events.



4.6.17.2 Triggering an "Expand" event

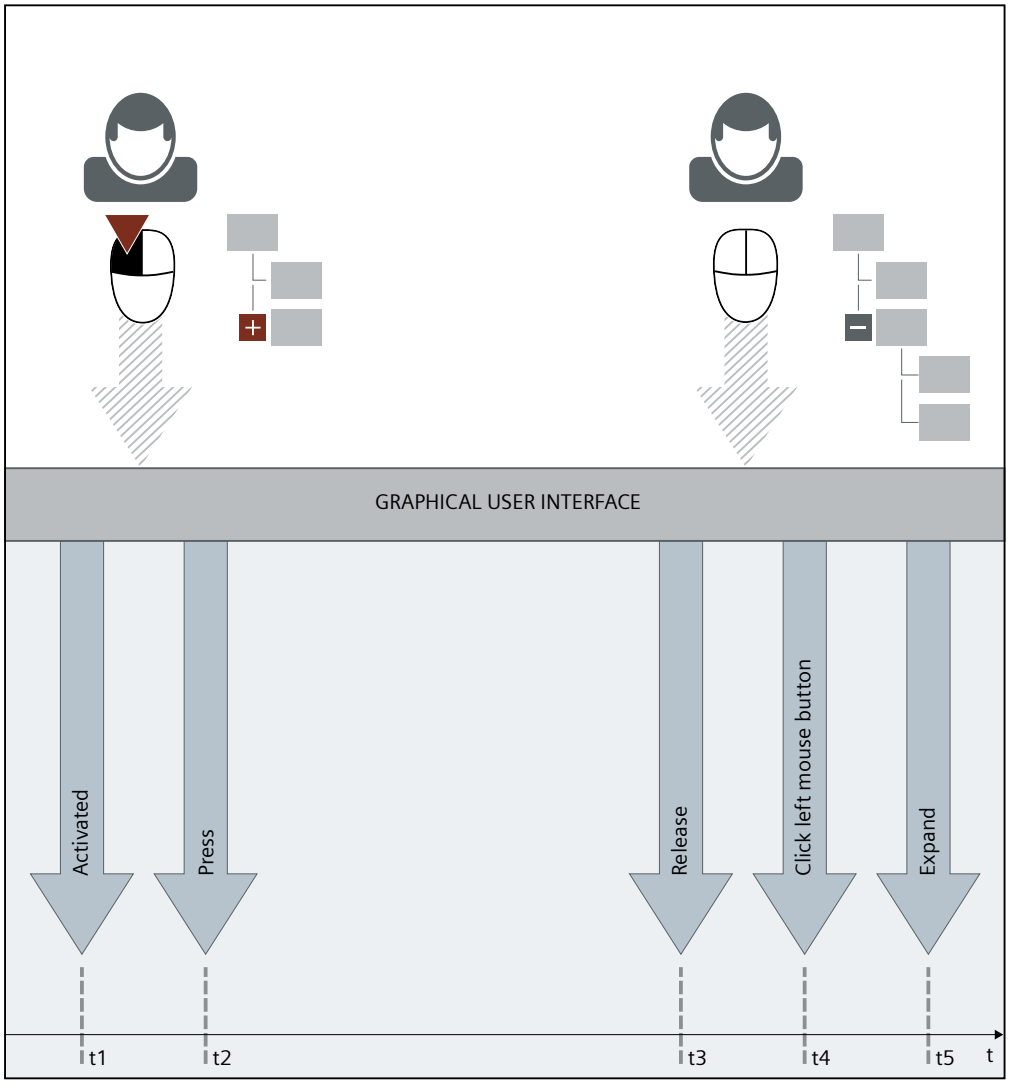
Introduction

You can trigger the "Expand" event in the "Plant overview" object by clicking the left mouse button.

Triggering an "Expand" event

If you want to trigger the "Expand" event by mouse operation, follow these steps:

1. Left-click on the symbol for "Expand".
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and finally "Expand" events.



4.6.17.3 Triggering an "Expand all" event

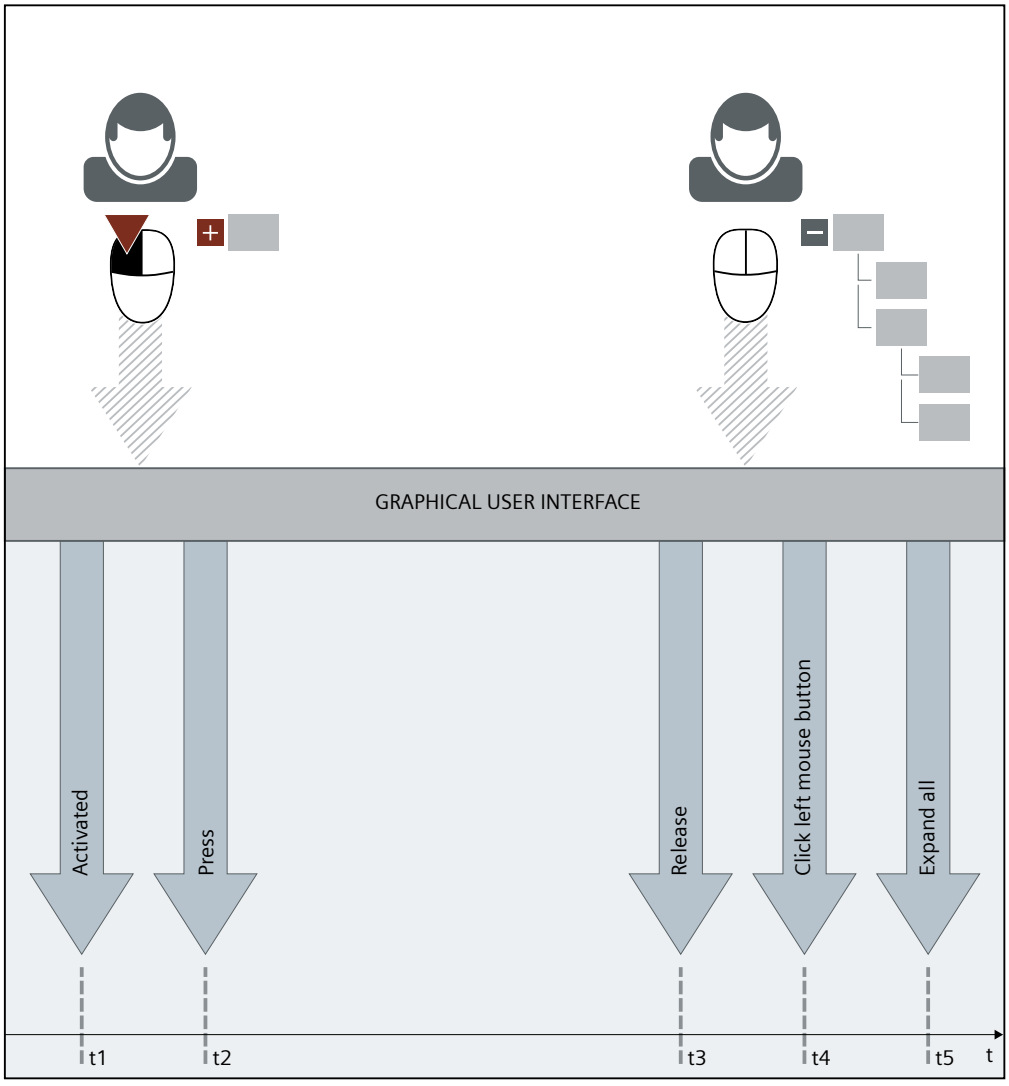
Introduction

You can trigger the "Expand all" event in the "Plant overview" object by clicking the left mouse button.

Triggering an "Expand all" event

If you want to trigger the "Expand all" event by mouse operation, follow these steps:

1. Left-click on the symbol for "Expand all".
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and finally "Expand all".



4.6.17.4 Triggering a "Minimize" event

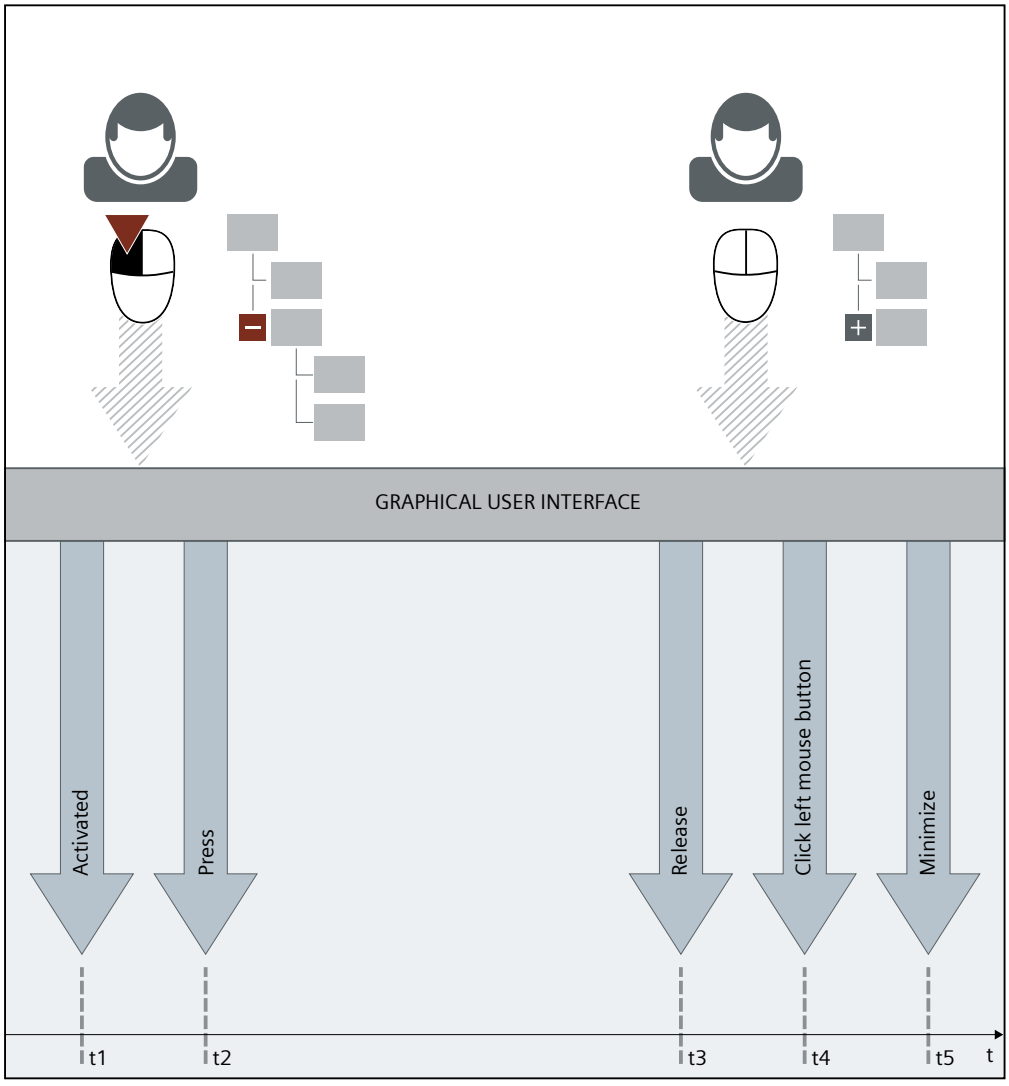
Introduction

You can trigger the "Minimize" event in the "Plant overview" object by clicking the left mouse button.

Triggering a "Minimize" event

If you want to trigger the "Minimize" event by mouse operation, follow these steps:

1. Left-click on the symbol for "Minimize".
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and finally "Minimize" events.



4.6.17.5 Triggering a "Minimize all" event

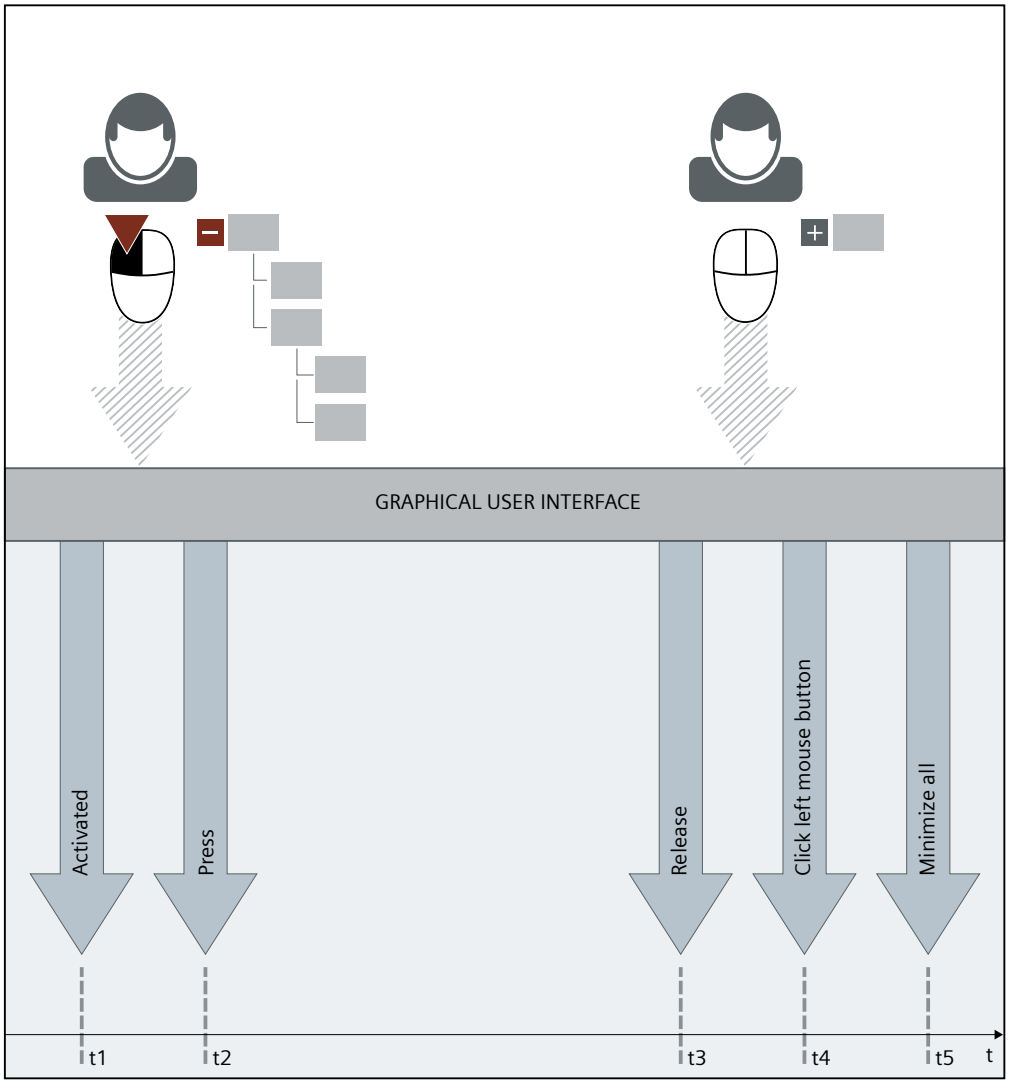
Introduction

You can trigger the "Minimize all" event in the "Plant overview" object by clicking the left mouse button.

Triggering a "Minimize all" event

If you want to trigger the "Release" event by mouse operation, follow these steps:

1. Left-click on the symbol for "Minimize all".
   Clicking triggers the events "Activated" and "Press".
2. Release the left mouse button.
   Releasing triggers the "Release", "Click left mouse button" and finally "Minimize all" events.



4.7 Configuring faceplates

4.7.1 Basics

4.7.1.1 Basics of faceplates

Introduction

Faceplates are user-defined groups of display and operating objects that are stored, managed and edited in the project library in a versioned manner. Faceplates are sometimes also referred to as "HMI blocks".

Faceplates support scripting and can therefore also open other faceplates in a pop-up window.

Depending on design and configuration, faceplates can be used universally and easily integrated into existing projects and employed several times.

Use

You use faceplates in order to create and re-use individually configured display and operating objects. You can edit faceplates centrally in the faceplate type. This reduces the configuration effort.

Depending on the application, a faceplate is a user-defined simple screen item or a detailed representation of a complex plant component.

Ideally, you should use faceplates for plant objects or parts that you use several times and that have identical data structures.

Note

Option to compile and load changes is lost

Please note the following instructions for compiling and loading changes:

- A dialog is often displayed when the option to compile only changes is about to be lost. The change can be confirmed or rejected.
  - If you confirm the change, the complete project must be compiled or loaded.
  - When you reject the change, the option to compile and load changes is retained.
- If you use the "Undo" button to undo a change that requires compiling or loading the entire project, the entire project must still be compiled or loaded.
- For the relevant changes and actions, an alarm is displayed in the Inspector window when the option to load changes is already lost. The entire project must be compiled and loaded.

Type/instance concept

Faceplates are based on a type/instance concept.

The faceplate type and its versions are managed centrally in the project library.

An instance of a version of a faceplate type is used in a screen as a faceplate container.

Faceplate type

- You create a faceplate type in the project library.
- More than one version of a faceplate type can be created.

Faceplate container

The faceplate container is an independent object in which a version of the faceplate type is instantiated.

- Each instance is connected to the faceplate version that has been used.
  This means that if you change a property or the data structure of a faceplate version, this property change immediately affects all faceplate instances that are based on the faceplate version.
- A faceplate container is used just like other display and operating objects in screens.
- If a version of a faceplate type has been instantiated in the container, the corresponding faceplate type is specified in the "Faceplate type of the instance" property.
- The tags and interface properties configured in the version of a faceplate type are linked in the faceplate container.

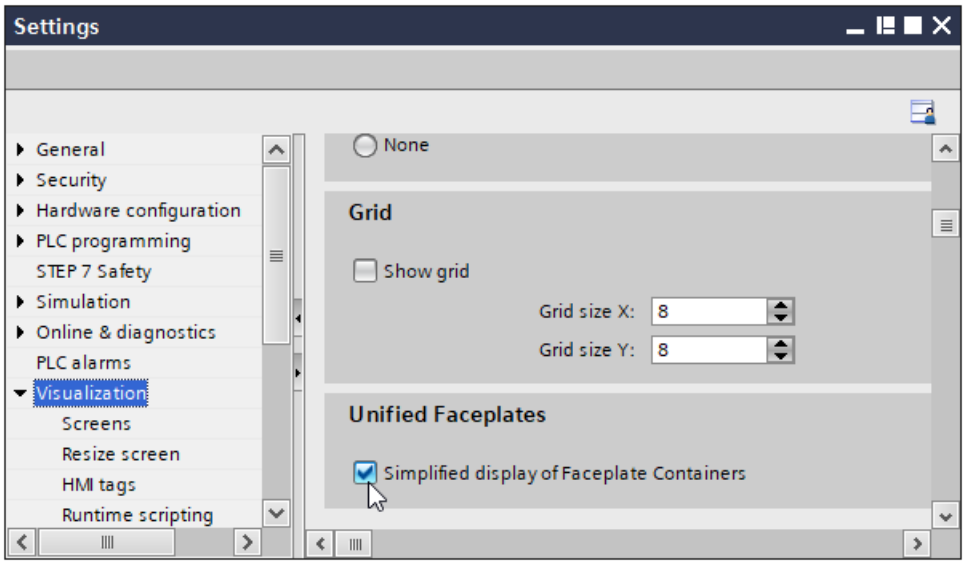
Example

If you use multiple valves within the project, you typically always use the same data structure to control and query the status of these valves. Therefore, it makes sense to use the same display and operating objects for the visualization of these valves.

1. In a faceplate type, you configure how the valve is displayed and which input and output tags the valve has in the form of tags.
2. If required, configure another faceplate type that contains the same data structure and functions as a pop-up window.
   This pop-up window can be called by the first faceplate type using a script.
3. For each valve in the system with the same data structure, instantiate the desired faceplate type and link its tags and PLC user data types with the corresponding valves in the system.

Improved performance when using a large number of faceplates

To improve performance during the engineering process when using numerous faceplates, activate the "Simplified display of Faceplate Containers" option under "Settings > Visualization > Unified Faceplate".



The faceplates are displayed in a simplified way when this option is activated in the engineering system. This change has no effect on the display in Runtime. In Runtime, faceplates are displayed in unrestricted quality.

4.7.1.2 Device dependency of faceplates

The functionality of the faceplate depends on the lowest device version of a faceplate type.

Devices

The following devices support faceplates:

- SIMATIC WinCC Unified PC
- SIMATIC Unified Comfort Panel

See also

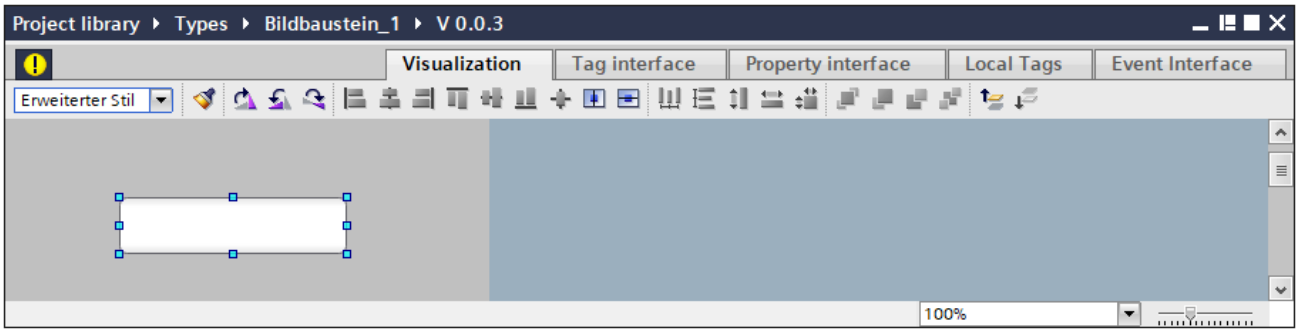
Lowest device version of a faceplate type (Page 529)

4.7.1.3 "Faceplate types" editor

Project library

You can create and edit faceplate types in the project library.

Layout



To view the tab of the editor, close the note .

Visualization

You can design the faceplate type in the "Visualization" tab.

You insert the following objects from the "Tools" task card:

- "Basic objects"
- "Elements"
- "Controls"
  The following controls are available:
  - Alarm control
  - Trend control
  - Faceplate container
    Empty container that you can later connect to a faceplate type.

You can find more information in the section Overview of screen objects (Page 269).

You can define the properties of the faceplate type and the objects in the Inspector window under "Properties > Properties". Here, you define display name, appearance and size, for example.

Assigning names to tags and properties

Note

Use unique names

The name of a tag or a property may be assigned across all tabs only once in a faceplate type.

Tags interface

On the "Tag interface" tab, you can configure the interface tags of the faceplate type and link the interface tags to HMI tags.

PLC user data types are supported on the interface.

Property interface

On the "Property interface" tab, you can configure the interface properties of the faceplate type.

The properties configured here are available for the instance of the faceplate type under "Miscellaneous > Interface".

You can create interface properties of the following data types:

- 64-bit integer
- Authorization
- Boolean
- Color
- Floating-point number
- Graphic
- Configuration string
- Multilingual text
- Resource list
- Unsigned 64-bit integer

Local tags

On the "Local Tags" tab, you can configure tags that are used exclusively within the faceplate type.

For example, you can use local tags in scripts within the faceplate type.

Event interface

In the "Event interface" tab, you configure events and associated parameters. You interconnect the events in the "Visualization" tab with the faceplate type or objects in the faceplate type.

On the faceplate container, the events are available under "Properties > Events". You can configure functions of the function list or scripts.

Interfaces of faceplate types can be copied

The entries of an interface in a faceplate type can be selected individually or copied in blocks as an entire interface and pasted again in another faceplate type.

This applies to:

- Tags interface
- Property interface
- Local tags
- Event interface

Pasting is also possible at the same place in the faceplate type.

To copy and paste, use the shortcut menu or the shortcut keys <Ctrl> + <C> and <Ctrl> + <V>.

To copy and paste all entries, select the entries by keeping the "Shift" key pressed or by using the keyboard shortcut <Ctrl> + <A> and copy via the shortcut menu or via the keyboard shortcut <Ctrl> + <C>. Insert the entries via the shortcut menu or the keyboard shortcut <Ctrl> + <V>.

4.7.1.4 Lowest device version of a faceplate type

Introduction

When you create the faceplate type, define the lowest device version for the faceplate type.

If you select the highest available device version, all functions are available. If you select a lower device version, some features are not available.

Note

Configured device version and lowest device version of the faceplate type

The configured device version of the HMI device in which an instance of the faceplate type is to be used must be equal to or higher than the lowest device version of the faceplate type.

The following changes in the project may cause error messages due to a device version that is too low:

- Changing the configured device version of the HMI device to a lower version
- Deleting a faceplate type
- Copying and pasting screens or faceplate containers between different HMI devices
- Copy screen with a faceplate container from the library to a screen

Note

TIA Portal version and device version of the faceplate type

A warning appears when you open a project that contains a faceplate type with a higher device version than the installed TIA Portal version in the project library or as an instance. It is possible to open the project, but the following restrictions apply:

- You cannot open or release the affected faceplate type.
- You cannot compile the project.
- If you open a screen that contains an instantiated faceplate type with a higher device version, the faceplate container is shown as empty.

Available functions depending on the device version

Function	Available as of version
Interface property of the "Resource list" data type	16.0.0.0
Interface property of the "Color" data type	16.0.0.0
Interface property of the "Configuration string" data type	17.0.0.0
Interface property of the "Authorization" data type	17.0.0.1 (Update 1)

Function	Available as of version
Interface property of the "Multilingual text" data type	18.0.0.0
Interface property of the "Graphic" data type	18.0.0.0
Alarm control	18.0.0.0
Trend control	18.0.0.0
Arrays from user data types	18.0.0.0
Arrays from values of simple data types	18.0.0.0
Link tags of the data type "DateTime" or "LTime" with the PLC tags.	18.0.0.0
Tags of the "PLCUDT" and "HMIUDT" data type	18.0.0.0
Using a faceplate type in another faceplate type	18.0.0.0
Interface events	18.0.0.0
Local tags	18.0.0.0
Using dynamic SVG graphics from the library in faceplates	18.0.0.0

4.7.1.5 Faceplates and TIA version upgrade

Version upgrade from V16 to V17 or higher

Note the following points when upgrading from version V16 to V17 or higher:

- As of V17, faceplates are located in the project library and are managed and used in types and versions.
- To prevent inconsistencies when linking data types, you must copy the user data types used in the PLC to the library before upgrading the version from V16 to V17 or higher.
  To do this, use drag-and-drop to move the PLC user data type to the library under "Project library > Types".
- When upgrading TIA Portal from V16 to V17 or higher, the names of existing faceplate types are automatically converted. In the higher version, the name of the faceplate type is given the version number "_0.0.1" as an extension. If, as a result, the permissible maximum total length of 128 characters is exceeded, an error message is displayed on the HMI device when the project is compiled.
  In this case, shorten the name of the faceplate type.
- The function rights "Function_right_01" to "Function_right_20" assigned to a faceplate in the V16 project are removed during the version upgrade.
- All master copies of faceplate types are removed during the version upgrade.
  Master copies of faceplate types are no longer supported as of V17.

4.7.2 Creating and managing faceplates

4.7.2.1 Creating a faceplate type in the project library

Introduction

Faceplate types are display and operating objects that are made up of several screen objects, such as I/O fields and controller blocks.

A faceplate type consists of one or more versions.

Requirement

The "Libraries" task card is open.

Procedure

1. Expand the "Types" folder in the project library.
2. Select the "Add new type" command.
   The "Add new type" dialog opens.
3. Select "HMI faceplate" and select "Unified Comfort Panel / WinCC Unified PC".
4. Enter a descriptive name in the "Name" field.

   Note

   The name must satisfy the following conditions:
   - Maximum character length: 128 characters
   - Unique name
   - No special characters:
     # $ +% . / : [ ] ' ~ " `
   - No JavaScript elements

5. Adapt the lowest device version.
   A new faceplate type with a preliminary faceplate version is created and shown in the project library.

Result

You have created a new faceplate type with a version. The preliminary faceplate version 0.0.1 is open in the editor and has the status "In progress".

See also

Basics of screens (Page 259)

4.7.2.2 Creating a faceplate type from a screen

As an alternative to creating a faceplate type in the project library, you can create a faceplate type directly from a screen of an HMI device. This can be useful, for example, if after configuring screen objects, you find that you want to reuse the screen objects and adapt tags and properties to the new application.

Note

- References to graphics in the "Graphic view" screen object are not transferred and must be inserted again.
- When tags are used for the dynamization, the reference is resolved as soon as a tag with the same name is created in the tag interface of the faceplate type.
- Configured function lists at events of screen objects are not transferred and have to be created again.

Note

Dynamic SVGs

If you create a faceplate type from a group of selected screen objects, dynamic SVG graphics are not included. To insert a dynamic SVG graphic into a faceplate, open the faceplate type in the editor and insert the SVG graphic from the library here.

Requirement

- A screen is configured and open.

Procedure

1. Select all the screen objects you want to use in the faceplate type by multiple selection in the screen.
2. Right-click to open the shortcut menu of the selected objects and select "Create faceplate". A new faceplate type is created in the project library that contains the copied screen objects and their configured properties.

Result

You have created a new faceplate type from a screen. The faceplate type contains the copied screen objects and their properties. The preliminary faceplate version is 0.0.1 and has the status "In progress".

4.7.2.3 Working with faceplate types and versions

Using the toolbar in the editor

Introduction

When you open the version of a faceplate type, the toolbar is displayed in the editor.

Toolbar

The toolbar provides information about the status of the opened version.

The toolbar allows access to the following functions, depending on the status of the opened version:

- Edit type
- Release version
- Discard changes and delete version
- Check if dependent types need to be adjusted.

Minimizing / showing the toolbar

To minimize the toolbar, click on or on "x".

To display the toolbar, click .

Editing a faceplate type

Introduction

When you open a released faceplate type for editing, a new version of the faceplate type is created based on the most recent version, "[Default]". When you edit an older version, the new version is based on the selected version.

Requirement

A faceplate type has been created.

Editing a faceplate type or version

You edit a released faceplate type or a type version as follows:

1. Select the faceplate type or the specific version you want to change.
2. Select "Edit type" from the shortcut menu.
   A new version of the faceplate type is created.
   The new version is opened in the "In progress" status.
   The new version is displayed in the editor.
3. Edit the newly created version.

Updating a faceplate type

Introduction

When a faceplate type is updated, all instances of the faceplate type are updated.

Requirement

- A faceplate type has been created and released.
- The "Libraries" task card is open.

Procedure

1. Select the faceplate type that you want to update.
2. Select "Update types" in the shortcut menu.
3. Select whether you want to update the types in a project or in the library.
   A dialog opens.
   - If "Project" is selected: Select the devices in which you want to perform the update.
   - If "Library" is selected: Select the library in which you want to perform the update.
4. Specify whether the unused type version is going to be deleted from the library.
5. Confirm your selection.

Note

Updating faceplate types in plant objects

Note that when faceplate types used in plant objects are updated, the interface assignment and dynamization within the plant object type is deleted if the interface tag or interface property was changed.

Result

You have updated all instances of the selected faceplate type in a library or in the project.

Renaming a faceplate type

Introduction

You can rename a faceplate type after it has been created. If you have referenced a version of a faceplate type in a script and you then rename the faceplate type, all references are automatically renamed.

Procedure

To enter a new name, select the faceplate type in the library.

1. Press <F2>.
   - or -
   In the shortcut menu, select "Rename".
2. Enter a new name.

The name is automatically checked for length, uniqueness and permitted characters. The name is reset to the previous name if the new name does not meet the requirements.

Opening a faceplate type or version write-protected

Introduction

You can open a faceplate type or an enabled type version write-protected. In write-protected mode, you can see all configurations within the type version but you cannot make any changes.

Requirement

At least one released type version is available in a library.

Procedure

To open a faceplate type in write-protected mode, select the faceplate type or a released version in the library.

When you open a faceplate type as read-only, the default version is always opened.

- Select "Open" from the shortcut menu.
  The toolbar is displayed and indicates that the faceplate type version is write-protected.

See also

Using the toolbar in the editor (Page 533)

Replacing a faceplate type

Introduction

The "Replace type" function replaces the used versions of a faceplate type in a project with a version of another faceplate type.

The source for replacing faceplate types is always a version. The target is always the enabled version of a faceplate type.

Requirement

- The "Libraries" task card is open.
- At least two faceplate types have been created and released.
- The faceplate type is used in the project.

Procedure

To replace all instances of the source type with the selected version of the target type, follow these steps:

1. Open the shortcut menu of the faceplate type or the version that you want to replace.
   If a faceplate type is selected for replacement, then the version defined as "Default" is used as the source version to be replaced.
2. Select "Replace type".
   The "Replace type" dialog is displayed.
   All compatible HMI devices and target types are displayed.
3. Select an HMI device.
4. Select a faceplate type.
5. Select a version of the target type.
6. Select "OK".
   A status message is displayed in the Inspector window in the "Info > General" tab.

Result

You have replaced the instantiated faceplate type with a different type.

Defining a version as the "default" version

When you add a faceplate type to a library, use types from a library, and release or update versions, the highest released version is used as the "default" version. You can specify another released version as the default version.

Requirements

- You have opened the project library or a global library.
- The desired version has been released.

Procedure

1. Select a version.
2. Open the shortcut menu.
3. Select "Set as "Default"".

Result

The newly set "Default" version is used instead of the highest released version when instantiating, creating, releasing and updating the type.

See also

Consistency status of types (Page 179)

Deleting a faceplate type or a version

Note the following when deleting faceplate types or versions:

- A faceplate type or a version can only be deleted if there are no dependencies on other types.
- When you delete a faceplate type, all versions of the type are deleted.
- When you delete all versions of a faceplate type, the type is also deleted.
- When you delete a version that has instances in the project, the instances are also deleted from the project.
- When you delete a type version with the "default" identifier and no additional versions of this type exist, the type is deleted.
- When you want to delete a type version with the "default" identifier and other versions of this type exist, the "default" identifier must first be assigned to a different version. Versions with the "default" identifier cannot be deleted if other versions of the type exist.
- If you delete a faceplate type or a version which exists in the "Types" folder in a global library, it remains in the global library.

Requirement

- The version that is to be deleted is released.
- The faceplate type that is to be deleted contains only released versions.

Procedure

1. To delete a faceplate version, select it and press <Del>.
   - or -
   Select "Delete" in the shortcut menu of the version.
   The "Confirm delete" dialog opens.
2. Confirm the delete operation.
   If the faceplate version is used in projects, the "Delete instances of the version" dialog appears, informing you that all instances of the version to be deleted will be removed.
3. Confirm the delete operation.

Note

If a deleted version is referenced in a script, an empty reference remains in the script after deletion. Remove it manually or insert a new reference.

Duplicating a faceplate type

Faceplate types in the project library can be duplicated. When you duplicate a faceplate type, the following applies to the duplicate:

- The duplicate is created in the same folder.
- The duplicate is created from the version of the type with the "default" identifier.
- The duplicate does not have an instance in the project.

Requirement

The faceplate type contains at least one released version.

Procedure

To duplicate a type in the project library, follow these steps:

1. Select the faceplate type or type version to duplicate.
2. Select "Duplicate type" from the shortcut menu.
   The "Duplicate type" dialog opens.

3. Enter the properties of the new type:
   - Enter a name for the new type in the "Name of type" field.
   - In the "Lowest device version" field, select the device version with which the type should be used.

     Note

     Only device versions that are the same as or higher than the original version can be selected.

   - Enter a version number for the new type in the "Version" field.
   - Enter the name of the editor who is responsible in the "Author" field.
   - Enter a comment on the type in the "Comment" field.
4. Confirm with "OK".
   The new faceplate type is generated with a released version.

Assigning a faceplate version number

A library is more clearly structured if types related by content have the same version number. The identical version number reflects the work progress. When you have completed the work on multiple associated faceplate types, you can assign the same version number to these types.

A log of the changes is created automatically. If you have versioned the faceplate types in the project library, you will find the log in the project tree under "Common data > Logs". If you have versioned the faceplate types in a global library, you will find the log in the "Common data > Logs" folder in the level below the global library.

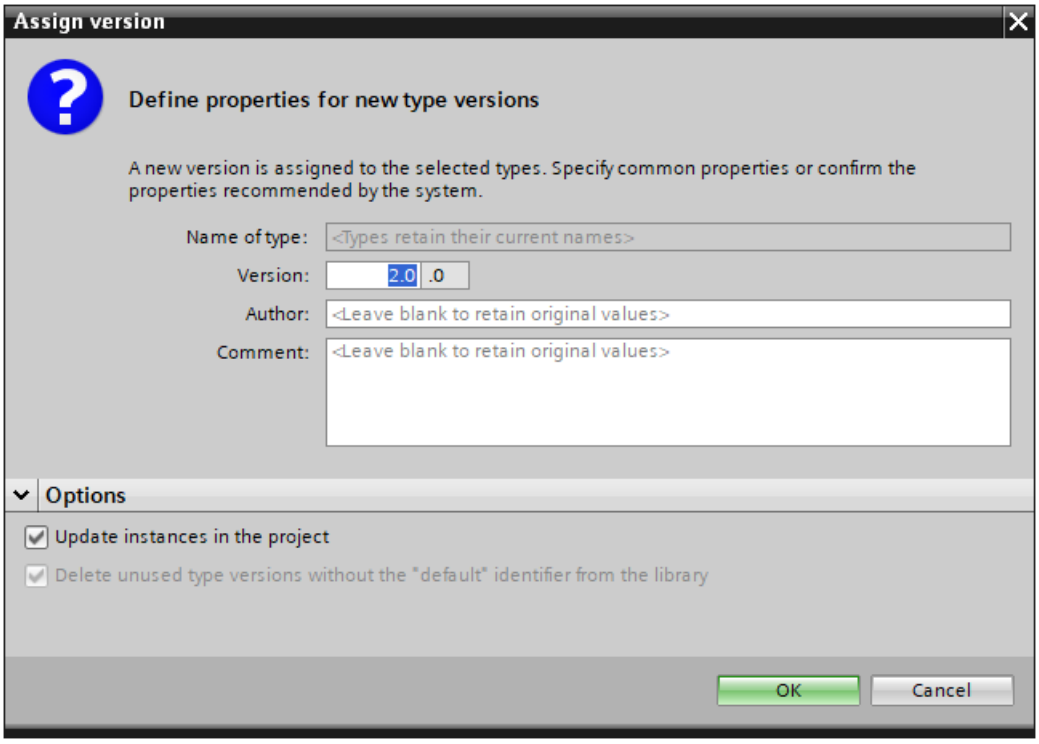
Requirement

- The "Libraries" task card is open.
- Faceplate types with "In progress" status have not been selected.

Procedure

To assign the same version to several faceplate types, follow these steps:

1. Select the faceplate types to which you want to assign a common version.
   Press and hold the <Ctrl> key and click on the faceplate types.
   If you have organized types in the library into folders, you can select one or more folders.
2. Select "Assign version" from the shortcut menu.
   The "Assign version" dialog opens.



3. If necessary, change the properties of the version:
   - In the "Version" field, determine the new version number. The version number must be higher than the highest version number of all selected types.
   - In the "Author" field, enter the person responsible for the version to be released.
   - In the "Comment" field, enter a comment on the version to be released.
4. Confirm with "OK".

Result

The selected versions of the faceplate types are changed as follows:

- A new version of all selected faceplate types is created with the specified version number.
- The properties are applied to all selected faceplate types. Lower versions used in the project remain unaffected by the changes. When you make no changes to the properties, the properties of the last released version or the version specified by the user as "default" of each faceplate type are applied.

- When a version is set as "default" by the user, the new version of the selected type is created from the default version with the specified version number. This newly created version will then have the "default" identifier.
- The version number of dependent types is incremented to the next free version number as long as the dependent types were not included in your selection. If you had selected a dependent type as well, the version number you specified will be assigned.

Note

Assigning a version number based on inconsistency

Updating the project library with versions of a global library or another TIA Portal instance can result in the project library having two faceplate versions with the same version number created by different authors. This leads to inconsistency and the project cannot be compiled. If this happens, change the version number of one of the duplicate versions.

See also

Consistency status of types (Page 179)

4.7.2.4 Editing the visualization of a faceplate type

You can edit the visualization on the "Visualization" tab of the editor.

The procedure corresponds to that for editing screens in the "Screens" editor.

Requirement

A faceplate type has been created.

Editing the visualization

1. Open a version of the faceplate type for editing.
   The visualization of the version is displayed on the "Visualization" tab in the editor.
2. Move objects from the "Toolbox" or "Libraries" task card to the "Visualization" tab of the faceplate version using drag-and-drop.
   Objects from the palettes "Controls" and "My Controls" are not available or only available to a limited extent.
   For dynamic SVG graphics, drag-and-drop pasting is only available as of version V18. If the device version is changed to a version < V18, you will get an error during the compile.

All the editing functions that you already know from configuring screens are available.

You can copy screen objects from one faceplate type to one or more other faceplate types without losing their attributes. If a copied property or tag does not exist in the target

faceplate type, the corresponding property or tag is highlighted in red and must be added in the target faceplate type.

Note

Screen objects can only be copied within a TIA Portal project. Copy operations between different TIA Portal instances are not supported.

Editing properties

Both the properties of the faceplate type and the properties of the objects used are edited in the Inspector window under "Properties > Properties".

1. Select the object.
   - If you want to adapt the properties of the faceplate type, click in a free area of the editor.
   - If you want to adapt the properties of a used object, click on it.
     The displayed handles indicate the selected object.
2. Open the shortcut menu and select "Properties".
   The Inspector window displays the properties of the object or faceplate type.
3. Edit the properties.

See also

Overview of screen objects (Page 269)

4.7.2.5 Configuring multilingualism for objects of a faceplate type

Introduction

The project languages are set in the "Project languages" editor. You specify which project language is to be the editing language and which the reference language.

Requirement

- A faceplate type has been created and opened for editing.
- At least one object is configured.

Procedure

1. Open the "Languages & Resources" menu command in the project tree.
   The lower-level elements are displayed.
2. Double-click on "Project languages".
   The possible project languages are displayed in the work area.

3. Enable the relevant project languages or disable the languages that you do not need.

Note

Copying multilingual objects

The copies of multilingual objects to a different project only include text objects in the project languages which are activated in the target project. Activate all project languages in the target project to include the corresponding text objects when transferring the copy.

Note

If you disable a project language, all text and graphic objects you have already created in this language are disabled from the current project. When the language is re-enabled, these are also re-enabled.

4. Go to the project library and open the faceplate type for whose objects you want to create multilingual texts.
5. Select the object for which you want to store a multilingual text.
6. Open the Inspector window under "Properties > Texts" and create the corresponding texts in the set languages.

Displaying multilingual texts in runtime

To activate the set languages in runtime, follow these steps:

1. Double-click on "Runtime settings" in the project tree.
2. Click on "Language & Font".
3. Enable the required languages.

See also

Exporting and importing library texts (Page 170)

4.7.2.6 Configuring tags in the faceplate type

Overview

Introduction

You configure interface tags or local tags in faceplate types. For each tag you define a data type. In addition to simple data types like Bool or Int, you can define arrays.

Tags of the data type "DateTime" or "LTime"

You can create tags of the data types "DateTime" or "LTime".

You can link interface tags of the "DateTime" or "LTime" data types, for example, with HMI tags of the data types "DateTime" or "LTime" on the faceplate container.

You can also link interface tags of the "DateTime" or "LTime" data types with the following data types of the S7-1500 PLC:

Data type of the interface tag	Data types of the PLC S7-1500
DateTime	Date
DateTime	Date_And_Time
DateTime	LDT
DateTime	DTL
LTime	Time
LTime	LTime
LTime	Time_Of_Day
LTime	LTime_Of_Day

Tags of the "Array" data type

You can define tags of the "Array" data type. The array index begins with 0.

Note

"WChar", "WString" and "HMIUDT"

Arrays of "WChar", "WString" and "HMIUDT" data types are not possible.

Configuring interface tags in the faceplate type

Introduction

In the faceplate type you can configure interface tags for dynamizing the properties of the objects contained in the faceplate type or embedding in scripts.

The interface tags of a faceplate type are linked exclusively via the faceplate container to the project tags.

Note

User data types

User data types, including with nested structures, are supported.

Note

Subsequent changes to interface tag names

Note that the values of interfaces that are already connected at the faceplate container are reset to their default values when you subsequently change names in the faceplate type.

Requirements

- The faceplate type has been created.
- The version is open for editing.
- The "Tag interface" tab is open in the editor.

Defining tags

Note

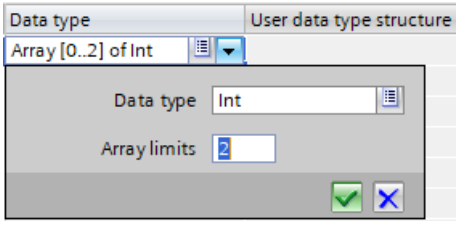
Only tags of the faceplate type are displayed within the editor.

1. Click on the button "Add Tag" or double-click the "Add" field.
2. Click on the tag name and assign a name. The name can be changed later.

   Note

   The tag name must satisfy the following conditions:
   - Maximum character length: 128 characters
   - Unique name
   - Beginning with a letter or underscore
   - No special characters:
     , ; . : ! ? " ' ^ ´ ` ~ - + = / \ ¦ @ * # $ % & § ° ( ) [ ] { } < >
   - No spaces
   - No JavaScript elements

3. Select the data type:
   - To select a simple data type, enter the name of the data type or select a data type from the selection list.
   - To select an array, click the arrow and select the data type.
     To determine the size of the array, enter a numerical value for the array upper limit.
     Example: 3 for an array with 4 elements [0 ... 3].
     To convert an array into a simple data type, delete the value for "Array limits".

     The WChar, Wstring and HMIUDT data types cannot be created as arrays.
   - To link a data structure, select "PLCUDT" for a PLC user data type or "HMIUDT" for an HMI user data type as the data type.
     Under "User data type structure", select a previously created user data type.

     Note

     User data types

     PLCUDT: You create PLC user data types on a PLC in project tree.

     HMIUDT: You create HMI user data types in the "Libraries" task card.

Changing tags

You can adapt the data type or the array limit of a tag retroactively.

To convert a simple data type into an array, select the arrow in the "Data type" column and specify the top array limit.

To convert an array into a simple data type, select the arrow in the "Data type" column and delete the top array limit.

Deleting an interface tag

To delete an interface tag, select the entry you wish to delete and click . Alternatively, delete the tag interface by pressing the <Del> key.

Changing the order of the interface tags in the editor

To change the order, select the respective entry and move it gradually up or down by clicking the buttons.

Result

You have configured the interface tags needed for the faceplate type.

The interface tags defined in the faceplate type are accessible in the corresponding faceplate instances and can be used for dynamization and for creating scripts within the faceplate type.

Configuring local tags in the faceplate type

Introduction

You can use local tags in a faceplate type to pass on information within the faceplate type. Elements of a faceplate can be dynamized, for example, as a function of the current properties of another element.

The local tags are not visible at an instance of the faceplate type and thus cannot be manipulated.

Requirements

- The faceplate type has been created.
- The version is open for editing.
- The "Local tags" tab is open in the editor.

Defining local tags

1. Double-click on the "<Add>" cell in the "Name" column.
2. If required, change the suggested name of the tag.

   Note

   The tag name must satisfy the following conditions:
   - Maximum character length: 128 characters
   - Unique name within the faceplate type
   - No special characters:
     : . # / % [ ] $ " ' * ? ~
3. Select the data type:
   - Simple data types: Enter the name of the data type or select a data type from the list.
   - Array: Select the "Array" data type. Click the arrow and define the data type and array limit.

Using a local tag

- To display a local tag and manipulate it in Runtime, connect the local tag, e.g. to an I/O field.
- To change the properties, e.g. color, of a screen object in Runtime as a function of the properties of another screen object, dynamize the properties using a local tag.
- If you use local tags in scripts, the events are assigned to the utilized screen objects.

Result

You have configured local tags for the faceplate type.

The local tags defined in the faceplate type are not accessible in the instances of the faceplate type.

4.7.2.7 Interface properties in faceplates

Overview

Introduction

You can define interface properties that you can use later in the faceplate type for the dynamization of properties. The following data types are available:

- 64-bit integer
- Authorization
- Boolean
- Color
- Floating-point number

- Graphic
- Configuration string
- Multilingual text
- Resource list
- Unsigned 64-bit integer

These data types can be instantiated in a faceplate container.

Note

Changing the names of interface properties

Note that the values of interfaces that are already connected at the faceplate container are reset to their default values when you subsequently change names in the faceplate type.

Description of the interface properties

Interface property	Description
64-bit integer	You can link properties that correspond to an integer with data type "64-bit integer". You can then assign corresponding elements with this numerical value within a screen in the faceplate instance. In this way you can assign different numerical values to objects in different faceplate instances.
Authorization	You can link properties of the "Authorization" type with authorizations. You can assign corresponding function rights in the faceplate instance. In this way, you can restrict the operator control of objects in different faceplate instances in different ways. If you specify the value "None" for the interface property on a faceplate instance, operator control is not restricted.
Boolean	You can link properties that correspond to a logical truth value (true and false) with data type "Boolean". You can assign corresponding elements with this value in the faceplate instance. In this way you can assign different values to objects in different faceplate instances.
Color	You can link properties of the type "Color" with the data type "Color". You can assign color values in the faceplate instance. In this way the objects can be displayed in different colors in different faceplate instances.
Floating-point number	You can link properties corresponding to a floating point number with the "Floating point number" data type. You can assign corresponding numerical values in the faceplate instance. In this way you can assign different numerical values to objects in different faceplate instances.
Graphic	You can associate properties that correspond to a graphic with the "Graphic" data type. You link graphics in the "Visualization" tab, for example, with the "Graphic view" and "Switch" screen objects. In the faceplate instance, you can assign graphics from the project's graphics collection or released type versions of the "Graphic" type from the project library. In this way you can assign different graphics to objects in different faceplate instances.
Configuration string	The data type "Configuration string" allows for the flexible assignment of values. A configuration string can include names or numbers, for example, that are addressed in scripts and can be transferred.

Interface property	Description
Multilingual text	You can link properties of the "Multilingual text" type with texts. In the faceplate instance, you can assign static texts, resource lists, tags or scripts. When entering static texts, use the key combination <Shift + Enter> to insert a line break. In the Inspector window you create texts for the projected languages under "Properties > Texts". In this way, you can configure different texts in different faceplate instances.
Resource list	You can link properties of the type "Resource list" with the "Resource list" data type. In the faceplate instance you can then assign elements from the graphic and text lists. In this way you can assign different text and graphic elements to objects in different faceplate instances.
Unsigned 64-bit integer	You can link properties that correspond to an unsigned 64-bit integer with the data type "Unsigned 64-bit integer". You can then assign corresponding numerical values in the faceplate instance. In this way you can assign different numerical values to objects in different faceplate instances.

Deleting interface properties

To delete an interface property, select the property you wish to delete and click .

Alternatively, press <Delete>.

See also

Creating a faceplate instance (Page 564)

Configure interface property

Requirement

- The faceplate type has been created.
- The version is open for editing.
- The "Property interface" tab is open.
- A screen is created in the HMI device.

Procedure

1. Click the "Add" field or click the button.
   A new interface property is created.
2. Change the values for the name, if required, and select the data type.

   Note

   The name must satisfy the following conditions:
   - Maximum character length: 128 characters
   - Unique name
   - Beginning with a letter or underscore
   - No special characters:
     , ; . : ! ? " ' ^ ´ ` ~ - + = / \ ¦ @ * # $ % & § ° ( ) [ ] { } < >
   - No spaces
   - No JavaScript elements
3. Switch to the "Visualization" tab.
4. Select the screen object that is to be linked to the interface property.
5. Open the "Properties > Properties" Inspector window.
6. Select the "Property interface" method in the "Dynamization" column.
7. Select the previously created interface property.
8. Release the version of the faceplate type and change to the project tree to interconnect the interfaces in the screen.
9. Open the screen and configure a faceplate container by dragging the faceplate type to the screen.
10. Open the Inspector window of the faceplate container and go to "Properties > Properties > Miscellaneous > Interface".
11. Select the created interface property. In the "Static value" column, assign a fixed value to the property. If you make several entries, separate the individual values with a semicolon.
    You can also assign values of the following categories to the property in the "Dynamization" column:
    - Tag
    - Script
    - Flashing (for colors)
    - Resource list (for text and graphic list elements)

    Note

    "Flashing" is not supported in Runtime.

	Tip for an efficient procedure
In the "Static value" column, you can copy and paste values via the shortcut menu of the text box.	

See also

Creating a faceplate instance (Page 564)

Example: Defining "Configuration string" and using it in a script

You can assign data values to interface properties with the "Configuration string" data type at the faceplate instance or using a script.

Requirement

- A faceplate type "Faceplate_1" has been created and opened for editing.
- A screen is created in the HMI device.

Note

Reference faceplate versions with full version numbers, such as: Faceplate_1_V_0_0_3.

Example of value assignment at the faceplate instance

1. Open the "Property interface" tab.
2. Create the following interface properties:
   - Label_Names
   - Label_Rotation
3. Assign the data type "Configuration string" to the interface properties.
4. Switch to the "Visualization" tab.
5. Configure three text boxes: "Text box_1", "Text box_2" and "Text box_3"
6. Configure a button with the text "Rotation" and one with the text "Read".
7. In the Inspector window of the "Rotation" button, open the "Events" tab.
8. Create the following script for the "Press" event:

```
export function Button1_OnDown(item, x, y, modifiers, trigger) {
   let myProperty = Faceplate.Properties.Label_Rotation;
   let angles = myProperty.split(";");
   Faceplate.Items("Text box_1").RotationAngle = angles[0];
   Faceplate.Items("Text box_2").RotationAngle = angles[1];
   Faceplate.Items("Text box_3").RotationAngle = angles[2];
}
```

9. In the Inspector window of the "Read" button, open the "Events" tab.

10. Create the following script for the "Press" event:

```
export function Button2_OnDown(item, x, y, modifiers, trigger) {
   let myProperty = Faceplate.Properties.Label_Names;
   let words = myProperty.split(";");
   Faceplate.Items("Text box_1").Text = words[0];
   Faceplate.Items("Text box_2").Text = words[1];
   Faceplate.Items("Text box_3").Text = words[2];
}
```

11. Release the faceplate type.
12. Change to the project tree.
13. Open a screen.
14. Create a faceplate instance by dragging the faceplate type from the project library to the screen.
15. Open the Inspector window of the faceplate instance and navigate to "Properties > Properties > Miscellaneous > Interface".
16. Navigate to the created interface properties and assign them meaningful values, such as:
    - Label_Names: "Label1;Label2;Label3"
    - Label_Rotation: "5;15;45"
17. Compile and load the project.

Result

When pressing the "Rotation" button, the text boxes of the faceplate are rotated by the values that were assigned to the interface property.

When pressing the "Read" button, the values assigned to the interface properties are read and the text boxes show the respective text.

Example of faceplate pop-up with value assignment via script

1. Open the "Property interface" tab.
2. Create the interface property "Title" and assign the "Configuration string" data type to it.
3. Switch to the "Visualization" tab.
4. Create a text box "Text box_1".
5. Open the Inspector window of the faceplate and switch to the "Events" tab.
6. Create the following script for the "Loaded" event:
   To do so, click on  "Convert function to script".

```
export function Faceplate_Typ_OnLoaded(item) {
     let myProperty = Faceplate.Properties.Title;
     Faceplate.Items("Text box_1").Text = myProperty;
}
```

7. Enable the faceplate.
8. Change to the project tree.

9. Open a screen.
10. Add a button "Button_1" on the screen.
11. Create a faceplate instance by dragging the faceplate to the screen.
12. Open the Inspector window of the button and switch to the "Events" tab.
13. Create the following script for the "Press left mouse button" event:
    To do so, click on "Convert function to script".

```
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    let data = {Title:"Text in Popup"};
    let po = UI.OpenFaceplateInPopup("Faceplate_1_V_0_0_1",
"Popup", data, UI.ActiveScreen, false);
    po.Left = 100;
    po.Top = 150;
}
```

Note

Referencing the faceplate type

The type is referenced by the name of the type, as shown in the properties, e.g. "Faceplate_1_V_0_0_1".

Include the complete version number. The version number specified in the script is automatically updated when a new version of the type is released.

14. Compile and load the project.

Result

When you press the button in runtime, the faceplate opens as a popup. The title "Text in pop-up" is read from the script and displayed in the text box.

See also

Creating a faceplate instance (Page 564)

4.7.2.8 Interface events in faceplates

Configuring an interface event in the faceplate type

Introduction

You use interface events to define events and associated parameters in the faceplate type. At the instance, you configure functions of the function list and scripts to the created event. This gives you the option of configuring various effects on the instances for an event defined in the faceplate type.

Various data types are available for the parameters associated with an interface event.

Requirement

- A screen is created in the HMI device.
- The faceplate type has been created.
- The version is open for editing.
- The "Event interface" tab is open.

Data types of parameters

The following data types are available for the parameters of interface events:

Data types	Description
Bool	Logical values (True/False)
Byte	Unsigned 8-bit value
Char	ASCII character
Color	Color
DateTime	Date/time information
DInt	Signed 32-bit value
DWord	Unsigned 32-bit value
HmiEventTrigger	The enumeration "HmiEventTrigger" can have the following values: • Unknown (0): Unknown • Touch (1): Triggered by touch HMI device • Left (16): Triggered by left mouse button. • Middle (17): Triggered by middle mouse button. • Right (18): Triggered by right mouse button. • Enter (256): Triggered by <Enter>. • Space (257): Triggered by <Space>. • Escape (258): Triggered by <Esc>.
HmiGesture	The enumeration "HmiGesture" can have the following values: • Unknown (0): unknown • SwipeLeft (1): Swipe left • SwipeRight (2): Swipe right • SwipeUp (3): Swipe up • SwipeDown (4): Swipe down
HmiKeyboardModifier	The enumeration "HmiKeyboardModifier" can have the following values: • None (0): None • Control (1): <Ctrl> • Shift (2): <Shift> • Alt (4): <Alt>
Int	Integer
LInt	Signed 32-bit value
LReal	64-bit floating-point number IEEE 754
LString	64-bit character sequence

Data types	Description
LWord	Unsigned 64-bit value
Real	32-bit floating-point number IEEE 754
SInt	Signed 8-bit value
String	32-bit character sequence
Time	Time information
UDInt	Unsigned 32-bit value
UInt	Unsigned integer
ULInt	Unsigned 64-bit value
USInt	Unsigned 8-bit value
Word	Unsigned 16-bit value

Configuring an interface event

1. On the "<Add>" or select the button  .
   A new interface event is created.
2. If required, change the values for the name.

   Note

   The name must satisfy the following conditions:
   - Maximum character length: 128 characters
   - Unique name
   - Beginning with a letter or underscore
   - No special characters:
     , ; . : ! ? " ' ^ ´ ` ~ - + = / \ ¦ @ * # $ % & § ° ( ) [ ] { } < >
   - No spaces
   - No JavaScript elements
3. Select "<Add>" below the interface event.
   A new parameter is created.
4. Change the value for the name.
5. Add more parameters if needed.
6. Switch to the "Visualization" tab.
7. Select the faceplate instance or a screen object with which the interface event is to be linked.
8. Open the "Properties > Events" Inspector window.
9. Select an event.
10. Click on  "Convert function to script".
    A script is created.
11. In the script, open the shortcut menu and select "Snippets > Faceplate > Raise a custom faceplate event".
    The following code is inserted:

```
let parameters = {Parameter_1:1, ColorParameter:0xff00ff00};
Faceplate.RaiseEvent("MyCustomEventName", parameters);
```

12. Adapt the names of the parameters in the script.
13. For the name of the interface event in the script, select the name that is created in "Event interface".
14. Release the type version of the faceplate.
15. Open the screen of the HMI device.
16. Create a faceplate instance of the released type version in the screen.
17. Select the faceplate instance.
18. Open the Inspector window under "Properties > Events".
19. Select the interface event in the event list.
20. Configure a function of the function list or a script to the event.

Editing an interface event

Note

Effects on the faceplate instance when the interface event is changed in the faceplate type

If you change the interface event, consider the following effects on the faceplate instance:

- If you have configured a script to the interface event and change the names of the parameters, you must manually update the parameters of the interface event used in it.
- If you have renamed the interface event, the name in the event list is updated automatically.
- If you have renamed the interface event or associated parameters and configured a script to the interface event, they are automatically updated in the script header.

1. Select the released type version.
2. Open the shortcut menu and select "Edit type".
   A new type version is created and opened for editing.
3. Select the "Event interface" tab.
4. Change the name of the interface event.
5. Change the name of the parameters.
6. Switch to the "Visualization" tab.
7. Select the faceplate instance or screen object linked with the interface event.
8. In the Inspector window, adapt the script under "Properties > Events":
   - Change the names of the parameters to match the changes in the "Event interface" tab.
   - Change the interface event names to match the changes in the "Event interface" tab.
9. Release the type version of the faceplate.
   Enable the "Update instances in project" option.
10. Confirm with "OK".
    The faceplate instance in the screen of the HMI device is updated.

11. If required, configure functions of the function list to the event again.
12. If required, adapt the names of the interface event parameters in the script.

Example: Configuring and using an interface event

With interface events, you define events in the faceplate type which you interconnect to instances with different functions of the function list or scripts.

Requirement

- A screen is created in the HMI device.
- A faceplate type "Faceplate_1" has been created and opened for editing.

Procedure

1. Switch to the "Event interface" tab of the faceplate type.
2. Click the "<Add>" field or select the  button.
   A new interface event is created.
3. Change the name of the interface event to "My_Interface_Event".
4. Select "<Add>" below the interface event.
   A new parameter is created.
   The data type of the parameter is "Int".
5. Change the name of the parameter to "Parameter_Int".
6. Select "<Add>" below the interface event.
   A new parameter is created.
7. Change the name of the parameter to "Parameter_Bool".
8. Change the data type of the parameter to "Bool".
9. Switch to the "Visualization" tab of the faceplate type.
10. Add a button.
11. Change the name of the button to "Button_1".
12. Select the "Button" screen object.
13. Open the Inspector window under "Properties > Events".
14. Create the following script for the "Click left mouse button" event:

```
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    let parameters = {Parameter_Int:90,Parameter_Bool:true };
    Faceplate.RaiseEvent("My_Interface_Event", parameters);
}
```

15. Release the type version of the faceplate.
16. Create the HMI tags "HMI_Tag_Int" of "Int" data type in the HMI device.
17. Create the HMI tags "HMI_Tag_Bool" of "Bool" data type in the HMI device.
18. Open the screen of the HMI device.

19. Create a faceplate instance of the released type version in the screen.
20. Select the faceplate instance.
21. Open the Inspector window under "Properties > Events" and select the event "My_Interface_Event".
22. Click on "Convert function to script".
23. Create the following script:

```
export function Faceplate_container_1_OnMy_Interface_Event(item,
Parameter_Int, Parameter_Bool) {
    let tag1 = Tags("HMI_Tag_Int");
    let tag2 = Tags("HMI_Tag_Bool");
    tag1.Write(Parameter_Int);
    tag2.Write(Parameter_Bool)
}
```

24. Configure 2 I/O fields in the screen:
    - "IO field_Int": Link the "Process value" property of the I/O field with the tag "HMI_Tag_Int".
    - "IO field_Bool": Link the "Process value" property of the I/O field with the tag "HMI_Tag_Bool".
25. Compile and load the project.

Result

The values of the IO fields change as follows when the "Button_1" button is pressed in Runtime:

- "IO field_Int": 90
- "IO field_Bool": 1

4.7.2.9 Checking the version consistency and fixing inconsistencies

Introduction

To ensure the consistency of a version of a faceplate type, you have two options:

- You can run a consistency check yourself before releasing a faceplate version.
- An automatic consistency check is performed when the version is released.

In both variants, the created version is checked for missing or faulty references to screens or tags, for example. A message is displayed in case of an error.

Requirement

- A faceplate version has been created but has not been released yet.
- Objects have been configured.

Checking the consistency yourself

1. Select the version whose consistency you want to check.
2. Select "Check consistency" from the shortcut menu.
   The result of the consistency check is displayed in the Inspector window in the "Info > General" tab.

Remedying the inconsistency

To remedy any inconsistency, follow these steps:

1. Open the Inspector window under "Information > General".
   An error and an error description are displayed in the Inspector window.
2. Click the green arrow in the "Go to" column.
   You are navigated automatically to the error location.
3. Eliminate the error.
4. If necessary, perform another consistency check to check whether all errors have been resolved.

You have fixed all consistency errors. The consistency check of the version of the faceplate type shows no errors.

See also

Consistency status of types (Page 179)

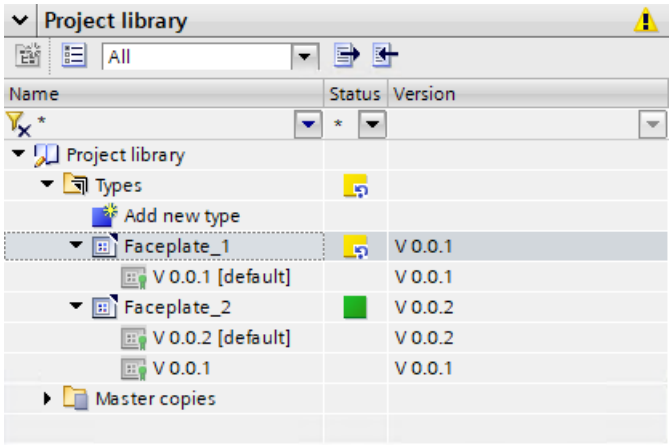
4.7.2.10 Checking the consistency at the faceplate type and fixing inconsistencies

Introduction

When editing versions of the faceplate types, incorrect referencing may unintentionally occur with the faceplate type and outdated instances if the default version of the dependent type is not used in the default version or a version other than the default version is instantiated in the device.

As soon as you release a version, a consistency check is automatically performed.

The "Status" column shows whether an inconsistency exists when referencing the faceplate type.



Resolving inconsistency through referencing of the non-default version of another faceplate type

Requirement

- A faceplate type has been created and released.
- The "Libraries" task card is open.
- The "Status" column in the project library displays the symbol .

Procedure

To correct an inconsistency which occurs through the referencing of a non-default version, follow these steps:

1. Open the shortcut menu of the inconsistent type.
2. Select the "Fix inconsistencies" menu item.
3. Select one of the following options:
   - "Adapt inconsistent type"
     A new version of the faceplate type is created and has the status "In progress".
     In the new version, the default version of the referenced faceplate type is automatically used.
   - "Set the currently referenced version as "default""
     The currently referenced version of the faceplate type is automatically set as the "default".

Remove inconsistency in the device through instantiating the non-default version

Requirement

- A faceplate type has been created and released.
- The "Libraries" task card is open.
- The "Status" column in the project library displays the symbol .

Procedure

To eliminate an inconsistency which occurs through the instantiation of a non-default version in the device, update the faceplate type (see section Updating a faceplate type (Page 534))

4.7.2.11 Releasing a faceplate version of a type

When you are finished editing a type version, release the version for productive use. Assign a version number for the release. You can also use multiple selection to release several versions at the same time.

If you have made structural changes to the type version to be released, such as changes at the interfaces, only the types that reference the changed type version and are affected by the change are set to the "In progress" status. To set all referencing types to the "In progress" status by default when they are released, select the check box "Set all dependent types to the 'In test' status" in the settings under "General > Library settings" in the "Release type" area.

Introduction

Only released versions of a faceplate type can be instantiated in a screen.

Requirement

- The "Libraries" task card is open or you are in the library view.
- The faceplate version that you want to release has the "In progress" status.
- The faceplate version is consistent.

Procedure

To release type versions, follow these steps:

1. Select the faceplate version you want to release.
2. Select the "Release version" command from the shortcut menu.
   The "Release type version" dialog box opens.
3. If necessary, change the properties of the version:
   - Enter a name for the faceplate type in the "Type name" field. If you have selected several versions for release, the "Name" field cannot be changed.
   - In the "Version" field, define a main and an intermediate version number for the version to be released. If you have selected several versions for release, the "Version" field cannot be changed and the last version number is used for the release.
   - In the "Author" field, enter the editor of the version to be released.
   - In the "Comment" field, enter a comment on the version to be released.

4. If necessary, change additional options of the faceplate version:
   - "Update instances in the project" option: Select the check box to update all instances in the project to the most recent version.
   - Option "Delete unused type versions without "Default" label from the library": Select the check box to delete all faceplate versions from the library that are not connected to any instance in the project. Versions with dependencies on other types are not deleted.
   - "Set dependent types to edit mode" option: If you have made incompatible changes to the type version to be released, such as changes at the interfaces, this check box is selected by default. Faceplate types that reference the changed type version are set to the "In progress" status by default. Clear the check box if you do not want to set the referencing types to the "In progress" status.
     If you have only made compatible changes to the type version to be released, this check box is cleared by default.
5. Confirm with "OK".

Result

- The selected faceplate version has been released.
- The properties are applied for the faceplate type itself, the version to be released, and for all future versions. Versions already released remain unaffected by the changes.
- The released faceplate version is given the "Default" identifier.
- If needed, all instances with the same original version are updated to the most recent version and the unused versions of the type are deleted.
- Depending on the changes you have made, releasing the type version has effects on types that reference this version:
  - If you have made incompatible changes to the type version to be released, such as changes at the interfaces, the types that directly reference the changed type version are set to the "In progress" state. The calling types still reference the last released version.
  - If you have only made compatible changes to the type version to be released, the types that directly reference the changed type version are not changed. The calling types reference the newly released version in this case.
- The icon in the "Status" column shows whether the references of the type are consistent with other types.

See also

Consistency status of types (Page 179)

Checking the version consistency and fixing inconsistencies (Page 559)

4.7.2.12 Creating a faceplate instance

Introduction

The faceplate type is stored in the project library. When you use the faceplate type in a screen, you create an instance of the faceplate type.

Note

Note that a faceplate is always configured for a particular class of HMI devices. For example, you cannot use a faceplate type that is configured for Runtime Advanced in a screen of a Unified HMI device.

Note

The number of faceplate instances in a screen is not limited. Note that the performance when opening or updating a screen is affected by the number of faceplate instances or the use of scripts in the faceplate instances.

Requirement

- A screen is open.
- The "Libraries" task card is open and the project library has been expanded.
- A faceplate type has been configured.
- The faceplate version you want to instantiate has been released.

Procedure using the project library

1. Drag the faceplate type or the desired version from the project library to the screen.
   The faceplate container with the faceplate instance is added to the screen.
   The version marked with "Default" is always used when you drag the faceplate type to the screen.
2. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface".
3. Connect the faceplate tags to project tags.
4. Specify the values of the interface properties.
5. Open the Inspector window under "Properties > Events".
6. Specify functions of the function list or scripts of the interface events.
7. Open the Inspector window under "Properties > Properties > Format > Fit to size".
8. To adjust the size of the faceplate or the container window, choose between "Fit window to screen" and "Fit screen to window".

   Note

   If you do not select resize, screen objects or navigation elements may not be displayed or may not be displayed completely.

Procedure using the "Faceplate container" control

1. Open the "Toolbox > Controls" task card.
2. Drag the "Faceplate container" control to the screen.
   An instance of the "Faceplate container" control is configured. No faceplate type is linked.
3. Open the Inspector window of the faceplate container under "Properties > Properties > Miscellaneous".
4. Select the desired faceplate type under "Faceplate type" in the "Static value" column.
5. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface".
6. Connect the faceplate tags to project tags.
7. Specify the values of the interface properties.
8. Open the Inspector window under "Properties > Events".
9. Specify functions of the function list or scripts of the interface events.
10. Open the Inspector window under "Properties > Properties > Format > Fit to size".
11. To adjust the size of the faceplate or the container window, choose between "Fit window to screen" and "Fit screen to window".

Note

If you do not select resize, screen objects or navigation elements may not be displayed or may not be displayed completely.

	**Tip for an efficient procedure**
In the "Static value" column, you can copy and paste values via the shortcut menu of the text box.	

Change the faceplate version in the container

You change the link to a faceplate version in the properties of the faceplate container under "Miscellaneous > Faceplate type". The selection window lists the released faceplate versions of all faceplates.

	**Tip for an efficient procedure**
• Under "Cross-references", you get a quick overview of all used objects within a faceplate container and the use of faceplate versions in screens. • You open the cross-references either in the Inspector window of the container under "Info > Cross-references" or via the shortcut menu of the respective object. • In the shortcut menu of the faceplate container, you can use the "Go to library version" function to jump directly to the referenced faceplate type in the project library.	

Result

- The faceplate type is instantiated in a faceplate container.
- The objects configured in the faceplate type are visible in the faceplate container.
- Interface tags and interface properties have been defined for the faceplate container.
- Functions of the function list and scripts are defined for the interface events of the faceplate container.
- If required, the properties of the faceplate container can be configured and dynamized in the inspector window.

4.7.2.13 Using a PLC user data type

Introduction

You can configure data blocks based on a PLC user data type (UDT). You can use tags that are based on this PLC user data type in faceplate instances.

The following advantages arise from reusing the PLC user data type:

- You minimize the configuration effort.
- You reduce the consumption of resources.
- They ensure unique and consistent naming of tags in data blocks and faceplate types and hereby significantly reduce the probability of configuration errors.

Note

Do not use more than one version of a PLC user data type in a faceplate type.

The following complex PLC data types are supported:

- User data types that are based on other user data types.
- User data types that use arrays comprising simple elements or user data types.

  Note

  Arrays whose limits are defined permanently with integer values or variably with user constants are supported.

- User data types that are based on a different user data type that, in turn, use arrays comprising simple elements or user data types.
- User data types that consist of further structured data types, e.g. CREF, PLCUDT, HMIUDT, NREF, IEC-specific parameters, and arrays comprising structured data types.

Note

Nesting is limited to 8 levels for structured user data types.

Depending on the size and nesting of the PLC user data types, the performance when assigning the PLC user data type and when opening the affected faceplate type may be impaired.

Requirement

- A SIMATIC S7-1200 or SIMATIC S7-1500 controller is configured.
- An HMI device has been configured.
- At least one PLC user data type has been configured.
- At least one PLC tag based on the PLC user data type has been configured.
- A faceplate type has been configured and opened for editing.
- A screen has been created.

Importing a device proxy into a project

If you import a device proxy into a project, the PLC user data types are not imported.

1. Create a user data type based on the PLC user data type in the project library of the source project.
2. Copy the user data type into the global library.
3. Apply the user data type to the project library of the project in which the HMI device is located.
4. Link the user data types with faceplate types as usual.

Procedure

1. Create a user data type that is based on the PLC user data type:
   - Drag-and-drop the PLC user data type from the device overview to the "Types" folder of the project library.
2. Create a PLC tag that is based on the PLC user data type:
   - Create a tag in the "PLC tags" editor.
   - Select a PLC user data type as the data type.

3. Connect an HMI tag to a PLC tag:
   - Create a tag in the "HMI tags" editor.
   - Under "Connection", select the external connection to the PLC.
   - Link the PLC tag that is based on a PLC user data type.
4. Create an interface tag in the faceplate type:
   - Go to the faceplate editor and open the "Tag interface" tab.
   - Create an interface tag.
   - Define the "PLCUDT" data type.
   - Under "User data type structure", select the PLC user data type.

   Alternatively, you drag-and-drop a version of the PLC user data type from the project library onto the "<Add>" field in the "Tag interface" tab.
   An interface tag with the data type "PLCUDT" is generated and linked with the version of the PLC user data type.
5. Release the faceplate type.
6. Create a faceplate instance:
   - Open the previously created screen and create an instance of the faceplate type.
   - Open the Inspector window of the faceplate instance.
   - Under "Properties > Properties > Miscellaneous > Interfaces", assign the HMI tags based on the PLC tags to the specific properties in the "Static value" column.

Note

Note that the version is not updated automatically in the faceplate type when you change the version of the referenced PLC user data type. Therefore, inconsistencies may occur if the linked version is not the default version. To clean up the inconsistency, you have the following options:

- In the shortcut menu of the inconsistent type, select the "Clean up inconsistencies" menu item. The type does not have to be released for this purpose.
- Open the faceplate version for editing and update the version of the linked PLC user data type manually in the task card "Tags interfaces" and enable the faceplate type.

When a PLC user data type is set to the "In test" state, the faceplate type referencing this PLC user data type is set to the "In progress" state. After release of the PLC user data type in the library, you may experience consistency problems in the faceplate type. In this case, update the PLC user data type reference in the tag interface to the current version.

Result

You are using tags that are based on a PLC user data type in a faceplate instance.

4.7.2.14 Using an HMI user data type

Introduction

You have the possibility to use HMI user data types in faceplates.

The use of HMI user data types provides the following advantages:

- You minimize the configuration effort.
- You reduce the consumption of resources.
- You ensure the unique and consistent naming of tags and thus significantly reduce the probability of configuring errors.

Requirement

- A SIMATIC S7-1200 or SIMATIC S7-1500 controller is configured.
- An HMI device has been configured.
- At least one HMI user data type has been configured.
- A faceplate type has been configured and opened for editing.
- A screen has been created.

Procedure

1. Create an interface tag in the faceplate type:
   - Go to the faceplate editor and open the "Tag interface" tab.
   - Create an interface tag.
   - Define the "HMIUDT" data type.
   - Under "User data type structure", select the HMI user data type.

   Alternatively, you drag-and-drop a version of an HMI user data type from the project library onto the "<Add>" field in the "Tag interface" tag. An interface tag with the data type "HMIUDT" is generated and linked with the version of the HMI user data type.
2. Create an object:
   - Drag an object, e.g. an I/O field, from the "Toolbox" task card to the "Visualization" tab.
   - Under "Properties > Properties > General > Process value", select "Tag" in the Dynamization field and assign an element to the interface tag.
   - Release the faceplate type.
3. Create a faceplate instance:
   - Open the previously created screen and create an instance of the faceplate type.
   - Open the Inspector window of the faceplate instance.
   - Assign the appropriate HMI tag under "Properties > Properties > Miscellaneous > Interfaces".

Result

You are using tags that are based on an HMI user data type in a faceplate instance.

4.7.2.15 Using a faceplate type in another faceplate type

Introduction

You can use a faceplate type in another faceplate type. In this way you assemble multiple faceplate types and increase the reusability of the faceplates.

The faceplate type used in another faceplate type is called an inner faceplate type. The faceplate type in which another faceplate type is used is called the outer faceplate type.

You can link the interface tags and interface properties configured in the inner faceplate type with interface tags and interface properties of the outer faceplate type. Local tags of the inner faceplate type cannot be linked to tags of the outer faceplate type.

Requirement

- A screen is created in the HMI device.

Procedure

Note

Reference to the same faceplate type is not supported

No other version of the same faceplate type can be used in a faceplate type.

1. Create a faceplate type in the library, e.g. "Faceplate_1".

   Note

   Smallest required device version

   The smallest required device version of the outer faceplate type must be equal to or larger than that of the inner faceplate type.

2. Configure interface tags and interface properties in the faceplate type, e.g. "Interface_Tag_Faceplate_1" and "Interface_Property_Faceplate_1".
3. Configure screen objects in the "Visualization" tab.
4. Link the screen objects with interface tags and interface properties.
5. Release the faceplate type.
6. Create another faceplate type, for example, "Faceplate_2".
7. Configure interface tags and interface properties in the faceplate type, e.g. "Interface_Tag_Faceplate_2" and "Interface_Property_Faceplate_2".
8. Switch to the "Visualization" tab of the outer faceplate type, e.g. "Faceplate_2".

9. Drag and drop the inner faceplate type, e.g. "Faceplate_1", from the library into the "Visualization" tab.
   Alternatively, you can configure the "Faceplate container" control and link the "Faceplate type" property to the inner faceplate type.
10. Select the inner faceplate type.
11. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface".
12. Link the interface tags and interface properties of the inner faceplate type or the faceplate container to the interface tags and interface properties of the outer faceplate type.
    If you use interface tags of the "PLCUDT" or "HMIUDT" data type, only compatible interface tags are displayed in the selection window.
13. Release the outer faceplate type.
14. Open the screen of the HMI device.
15. Drag and drop the outer faceplate into the screen.
16. Connect the faceplate tags to project tags.
17. Specify the values of the interface properties.
18. Open the Inspector window under "Properties > Events".
19. Specify system functions of the function list or scripts of the interface events.

4.7.2.16 Copying faceplate types and faceplate instances to other projects

Introduction

Faceplate types can be transferred to other projects.

Requirements

- The target project contains the devices on which faceplates can be used.
- Faceplate types: If PLC user data types are used in the faceplate type, the same PLC user data types must be available in the target project.
- Faceplate instances: It must also be possible to integrate the tags of the used faceplate types into the target project.
- Both projects (source and target) are open in different instances of TIA Portal.

Note

Dependencies

Hierarchical dependencies exist between user data types, faceplate type and faceplate instances:

1. Faceplate instances use faceplate types.
2. Faceplate types use user data types where necessary.

Therefore, note the order:

1. Configure PLC user data types.
2. Copy the faceplate type.
3. Copy the faceplate instances.

Configuring PLC user data types

1. Switch to the project from which you want to copy the faceplate type.
2. Check the faceplate type to be copied for any used PLC user data types.
3. Go to the target project.
4. In the target project, configure the PLC user data types required in the faceplate type that is to be copied.

Copying a faceplate type

1. Switch to the project from which you want to copy the faceplate type.
2. Select the desired faceplate type.
3. Copy the desired faceplate type.
4. Go to the target project.
5. Select the "Types" folder in the project library.
6. Insert the faceplate type into the target project.
7. Integrate the required PLC user data types into the new faceplate type.

Copying via "Global libraries"

Alternatively, you can make the faceplate type available for other projects by copying it to a global library.

To copy a faceplate type to another project via a global library, follow these steps:

1. Switch to the project from which you want to copy the faceplate type.
2. Copy the faceplate type.
3. Open the "Global libraries" pane and paste the copied faceplate type there.
4. Go to the target project.

5. Open the "Global libraries" pane.
6. Copy the previously pasted faceplate type from the global library to the project library.

Note

Name redundancy

In case of a name redundancy, the name of the copied faceplate type is given a version number in the form "-[number]", e.g. "Faceplate_1".

7. Integrate the required PLC user data types into the new faceplate type.

Copying a faceplate instance

1. Switch to the project from which you want to copy the faceplate instance.
2. Select the desired faceplate instance.
3. Copy the desired faceplate instance.
4. Go to the target project.
5. Open the screen in which the faceplate instance is to be inserted.
6. Paste the faceplate container.
   The faceplate type is created in the project.
7. Link the required tags in the faceplate container with those in the project.

See also

Basics of faceplates (Page 524)

4.7.3 Connecting faceplate types to OPC UA

OPC UA is a standardized manufacturer-independent software interface for data exchange in automation engineering.

You can use data values from an OPC UA connection in faceplate types.

Requirement

- A OPC UA connection is configured.

Procedure

To use data values from an OPC UA connection, you must assign the data types to the corresponding Unified data types. You can find the assignment of OPC UA and Unified data types in the table below:

OPC UA data type	Unified Faceplates data type
Int32	DInt
Boolean	Bool
Byte	USInt
DateTime	DateTime
Double	LReal
Float	Real
Int16	Int
SByte	SInt
String	WString*
UInt16	UInt
UInt32	UDInt32

*Only possible as a local tag, data type for interface tags not available.

Data types that are not listed in the table are not supported by Unified Faceplates.

For more information on OPC UA, refer to the Runtime - Open Platform Communications (OPC) documentation.

4.7.4 Dynamizing faceplates

4.7.4.1 Basics for the dynamization of faceplates

General

Both the properties of the faceplate type and the properties of the objects used in the "Visualization" tab are dynamized in the Inspector window under "Properties > Properties".

You link functions of the function list and scripts to the faceplate type or objects of the faceplate type in the "Visualization" tab in the Inspector window under "Properties > Events".

Use

You can dynamize events and properties of faceplates at two levels.

1. Dynamizing a faceplate type
2. Dynamizing a faceplate instance

Dynamizing a faceplate type

Note

Using tags

Only use tags that are defined within the faceplate type.

You can dynamize properties of objects or events in the faceplate type on the "Visualization" tab of the "Faceplate types" editor. You configure the individual objects as in the "Screens" editor.

- To dynamize properties, the following methods are available in the editor:
  - Tag
  - Script
  - Flashing (for colors)

    Note

    "Flashing" is not supported in Runtime.

  - Property interface
    With this method, you use the interface properties configured in the "Property interface" tab.

  Depending on the property, only certain methods are available.
- You dynamize events by using functions of the function list or scripts.
  Faceplates support various methods. For more information on supported methods, see the AUTOHOTSPOT section.
- You do not have access to the tags and scripts of the project within the faceplate type. You must therefore configure interface tags in the faceplate type that you link with the tags of the project in the faceplate instance.
- Each faceplate instance created with the faceplate version has the same objects with identical dynamization.
  You can edit this dynamization exclusively in the "Faceplate types" editor.

Dynamizing a faceplate instance

You configure the events or dynamic properties individually on the faceplate container. This dynamization refers exclusively to the faceplate container.

Properties of the objects used in the faceplate type cannot be dynamized directly. For this purpose, interface tags, interface properties or interface events must be defined in the faceplate type, via which a dynamization is triggered.

See also

Basics of dynamizing screens (Page 462)

4.7.4.2 Dynamizing a faceplate type

Introduction

You dynamize screen objects in the faceplate type in the Inspector window editor.

To trigger the reaction to events in the faceplate instance, you can, for example, use the interface tags that were created beforehand in the faceplate type.

You can connect the dynamic properties of the faceplate container with a tag or a script that provides the property with values in Runtime.

Creating scripts in the faceplate type works the same way as creating scripts on objects in screens.

Support of system functions

The use of system functions within faceplates is not fully supported. The following system functions are available in the function list and in scripts:

Function list		Script
Alarm system		
	CreateOperatorInputInformation	HMIRuntime.Alarming.SysFct.CreateOperator-InputInformation
	CreateSystemInformation	HMIRuntime.Alarming.SysFct.CreateSystemIn-formation
	CreateSystemAlarm	HMIRuntime.Alarming.SysFct.CreateSystemA-larm
Resource		
	LookUpText	HMIRuntime.Resources.SysFct.LookUpText
Tag		
	UpdateTag	HMIRuntime.Tags.SysFct.UpdateTag
	IncreaseTag	HMIRuntime.Tags.SysFct.IncreaseTag
	InvertBitInTag	HMIRuntime.Tags.SysFct.InvertBitInTag
	ResetBitInTag	HMIRuntime.Tags.SysFct.ResetBitInTag
	ShiftAndMask	HMIRuntime.Tags.SysFct.ShiftAndMask
	SetBitInTag	HMIRuntime.Tags.SysFct.SetBitInTag
	SetTagValue	HMIRuntime.Tags.SysFct.SetTagValue
	DecreaseTag	HMIRuntime.Tags.SysFct.DecreaseTag

The available system functions can be displayed grouped by areas or sorted alphabetically.

You can find more information at AUTOHOTSPOT.

Requirement

A faceplate type has been created.

Dynamizing events

Note

Using tags

Use tags that have been defined within the faceplate type under "Tag interface" or "Local tags".

You link the tags defined in the faceplate type with the project tags in the faceplate instance.

1. Select the faceplate type or an object in the faceplate type.
2. Open the Inspector window under "Properties > Events".
3. Select an event.

   Note

   "Cleared" event

   The "Cleared" event is triggered when the faceplate is already resolved. Therefore, access to interface tags and local tags of the faceplate type is not possible.

4. Select a function from the function list or create a script.

Note

Global definition in the faceplate type

All definitions in the "Global definition" area of the "Scripts" editor in the faceplate type are used in all instances of the faceplate type. When you define a tag in the "Global definition" area, for example, any value changes of this tag can be seen in all instances of the faceplate type.

Tip for an efficient procedure
You are supported by snippets when creating scripts; you access these snippets from the shortcut menu under "Snippets > Faceplates".

Dynamizing object properties

1. Select the relevant object.
2. Open the Inspector window of the object and select "Properties".

3. In the "Dynamization" column of the Inspector window, select the menu of the property that you want to dynamize.
4. Select the method:
   - Tag
   - Script
   - Property interface
   - Flashing (for colors)

Note

The availability of methods is dependent on the selected property.

"Flashing" is not supported in Runtime.

See also

Introduction to runtime scripting (Page 969)

System functions (Page 909)

4.7.4.3 Dynamizing a faceplate instance

Introduction

You dynamize properties of the faceplate instance in exactly the same way as you dynamize properties of another object in the "Screens" editor.

In the "Screens" editor, you can connect the dynamic properties of the faceplate container with a tag or a script that provides the property with values in Runtime.

You have previously created the tags and scripts in the project.

In the Inspector window under "Properties > Events" you will find the interface events defined in the faceplate type.

Requirement

A faceplate container with a faceplate instance is inserted in the screen.

Note

Requirement for triggering the "Enabled" event in Runtime

To trigger the "Enabled" event at the faceplate container, you must enable the options "Display frame" and "Show heading" in the properties of the control under "Window settings".

Dynamizing events

1. Select the faceplate container.
2. Open the Inspector window under "Properties > Events".
3. Select an event.
4. Select a function from the function list or create a script.

	**Tip for an efficient procedure**
You are supported by snippets when creating scripts; you access these snippets from the shortcut menu under "Snippets > Faceplates".	

Dynamizing object properties

1. Select the faceplate instance.
2. Open "Properties" > "Properties" in the Inspector window of the faceplate instance.
3. In the "Dynamization" column, select the menu of the property you want to dynamize.
4. Select the required method:
   - Tag
   - Script
   - Resource list (for strings)
   - Flashing (for colors)

   Note

   "Flashing" is not supported in Runtime.

   Depending on the property, only certain methods are available.

See also

Basics for the dynamization of faceplates (Page 574)

Basics of dynamizing screens (Page 462)

4.7.4.4 Accessing properties of the faceplate container with a script

Procedure

To read or write faceplate container properties from a faceplate type, use the `Parent` property of the `Faceplate` object in a script.

You can access the following properties of the faceplate container, for example:

- Position
- Height and width

- Visibility
- Interface tags
- Interface properties

Example: Reading the position of the faceplate container

```
HMIRuntime.Trace(Faceplate.Parent.Top);
HMIRuntime.Trace(Faceplate.Parent.Left);
```

Read out the alarm with the RTIL Trace Viewer.

Example: Changing the position of the faceplate container via a button

```
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
// Read Faceplate container coordinates
   let containerTop = Faceplate.Parent.Top;
   let containerLeft = Faceplate.Parent.Left;

// Write Faceplate container coordinates. Increment with 100 pixels
   Faceplate.Parent.Top = containerTop + 100;
   Faceplate.Parent.Left = containerLeft + 100;
}
```

Example: Read connected tags

```
let TagNameSystem;
TagNameSystem = Faceplate.Parent.Properties.TagProperty_1.Tag;
HMIRuntime.Trace("System Tag Name: " + TagNameSystem);
```

Read out the alarm with the RTIL Trace Viewer.

See also

Tracing with the RTIL Trace Viewer (Page 1023)

4.7.4.5 Configure faceplate as pop-up

Introduction

A faceplate can be opened in Runtime as pop-up window by means of a script.

Note

Only faceplate versions used in the HMI device are available in Runtime.

- Use the faceplate version in the HMI device as faceplate container or in a script as reference. The reference is not resolved if you dynamically generate the name of the faceplate version to be opened as a popup with a script.
- If there is no faceplate version with the name, an empty container is opened.

You have the option of calling a pop-up window either inside or outside a faceplate type:

- Outside a faceplate type: You transfer all interface tags and interface properties in the script.
- Inside a faceplate type: The interface tags and interface properties of the pop-up are applied from the faceplate type in which the pop-up window is triggered.

	**Tip for an efficient procedure**
The "OpenFaceplateInPopup" and "Close" methods are available as a snippet. The available snippets are displayed in the shortcut menu of the "Scripts" editor.	

Requirement

- A screen is open.

Open and close pop-up windows outside a faceplate type

To configure a faceplate as a pop-up window, for example, to an event of a screen object, follow these steps:

1. Create a faceplate type.
2. Create faceplate tags, for example, "Interface_Tag_1".
3. Create interface properties, for example, "Color_Property_1" (color data type) and "ResourceList_Property_1" (resource list data type).

   Note

   Interface property of the "Multilingual text" data type

   Interface properties of the "Multilingual text" data type cannot be passed to a faceplate that is called as a popup window outside of a faceplate type.

4. Configure the visualization of the faceplate.
5. Interconnect the tags and properties.
6. Release the version of the faceplate type.

7. Open the screen of the HMI device.
8. Configure the screen object that is to trigger the event, for example, a button.
9. Open the Inspector window under "Properties > Events".
10. Select an event.
11. Click on "Convert function to script".
    A function is generated.
12. Assign tags and interface properties to the faceplate interface. In the example, the "data" parameter is used.
    To specify text lists, use the prefix "@Default", for example, "@Default.Text_list_1".

    Note

    Graphic lists are not supported.

13. Create a global definition for the faceplate pop-up.
    Example:
    ```
    //JEx: "Faceplate in Popup"
    var po;
    ```
14. Use the "OpenFaceplateInPopup" method.
    - Define the type of the pop-up window, for example, "Faceplate_1".

      Note

      The version of the faceplate type preset as "Default" is automatically displayed in Runtime.

      Note

      Note that you must replace spaces and dots in the faceplate name with underscores when referencing faceplate versions.

      Reference faceplate versions with full version numbers, such as: Faceplate_1_V_0_0_3.

    - Define the title of the pop-up window, for example, "Popup".
    - Specify the parameter of the interface, for example, "data".
    - Optional: The pop-up window is closed automatically when the specified screen is exited. Default: 0. The pop-up window remains open until it is closed manually or Runtime is terminated.
      In the example, the pop-up window is closed as soon as the active screen is exited.
    - Optional: Specify whether the pop-up window is hidden.
      Default: false. The pop-up window is visible.
15. If required, specify where the pop-up window is opened.
16. Configure a "Close pop-up" button.
17. Open the Inspector window of the button under "Properties > Events".
18. Select an event.
19. Click on "Convert function to script".
    A function is generated.
20. Use the "Close" method.

Example code for calling the faceplate pop-up outside a faceplate type

```
//JEx: "Faceplate in Popup"
//TagsRequired: "HMI_Tag_1"
//FPlateRequired: "Faceplate_1", "Interface_Tag_1", "Color_Property_1",
"ResourceList_Property_1"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
let data = {Interface_Tag_1:
{Tag:"HMI_Tag_1"},Color_Property_1:0xff00ff00,ResourceList_Property_1:"@De
fault.Text_list_1"};
po = UI.OpenFaceplateInPopup("Faceplate01_V_0_0_3", "Popup", data,
UI.ActiveScreen, false);
po.Left = 100;
po.Top = 150;
}
```

Example code for closing the faceplate pop-up outside a faceplate type

```
//JEx: "Close Faceplate in Popup"
//FPlateRequired: "Faceplate01_V_0_0_3", "po"
export function Button_2_OnTapped(item, x, y, modifiers, trigger) {
if (po)
{
po.Close();
po = undefined;
}
}
```

Calling a pop-up window within a faceplate type

To configure a faceplate as a pop-up window, for example, to an event of a screen object within a different faceplate type, follow these steps:

1. Create a faceplate type, for example, "Faceplate_1".
   The event that opens the pop-up window is configured in this faceplate type.
2. Create another faceplate type, for example, "Popup_1".
   This faceplate type is opened as a pop-up window.
3. Create the same interface tags in both faceplate types, for example, "Interface_Tag_1".
4. Create the same interface properties in both faceplate types, for example, "Color_Property_1" and "ResourceList_Property_1" (resource list data type).

   Note

   The name and data type of the interface tags and interface properties must match. The order may be changed. Missing interface tags or interface properties are not transferred.

5. Configure a screen object that is to open the pop-up window, e.g. a button, in the "Visualization" tab of "Faceplate_1".
6. Configure additional objects to visualize the two faceplate types.
7. Interconnect the tags and properties of both faceplate types.

8. Select the screen object which is to open the pop-up window.
9. Open the Inspector window under "Properties > Events".
10. Select an event.
11. Click on "Convert function to script".
    A function is generated.
12. Use the "OpenFaceplateInPopup" method.
    - Define the type of the popup window with the desired version, for example, "Popup_1_V_0_0_1".
      The correctly detected reference to the faceplate type is highlighted in green.
    - Define the title of the pop-up window, for example, "Popup".
    - Optional: The pop-up window is closed automatically when the specified screen is exited.
      Default: `independentWindow = false`.
      The pop-up window is closed when the faceplate or the screen is exited.
      In the example, the pop-up window remains open until the pop-up window is closed manually or Runtime is exited.
    - Optional: Specify whether the pop-up window is hidden.
      Default: `invisible = false`.
      The pop-up window is visible.
13. If required, specify where the pop-up window is opened.
    ```
    po.Left = 100;
    po.Top = 150;
    ```
14. Switch to the faceplate type "Popup_1".
15. Configure a "Close pop-up" button.
16. Open the Inspector window of the button under "Properties > Events".
17. Select an event.
18. Select the "Convert function to script" button.
    A function is generated.
19. Use the "Close" method.
20. Release both type versions.

Example code for calling the faceplate pop-up within a faceplate type

```
//JEx: "Faceplate in Popup within another Faceplate"
//FPlateRequired: "Popup_1"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
po = Faceplate.OpenFaceplateInPopup("Popup_1_V_0_0_1", "Popup", true,
false);
po.Left = 100;
po.Top = 150;
}
```

Example code for closing the faceplate pop-up within a faceplate type

```
//JEx: " Close Faceplate in Popup within another Faceplate"
//FPlateRequired: "Popup_1_V_0_0_1"
export function Button_2_OnTapped(item, x, y, modifiers, trigger) {
Faceplate.Close();
}
```

Result

When the configured event is triggered in Runtime, the pop-up window opens or closes.

4.7.5 Example: Creating and using faceplates

4.7.5.1 Example: Configuring faceplates

Procedures overview

The example is divided into the following steps:

1. Create HMI tags.
2. Configure faceplate types:
   - A faceplate type that is instantiated in the screen.
   - A faceplate type that is opened as a pop-up from within the faceplate.
   - A faceplate type that is instantiated in the faceplate.
3. Configure interface tags in the faceplate types.
4. Configure local tags in faceplate types.
   Instead of tags: Use PLC user data types and thus minimize configuration work.
5. Instantiate inner faceplate type in outer faceplate type,
6. Configure interface properties in faceplate types.
7. Create an interface event.
8. Create a script to modify tags.
9. Creating the local scripts,
10. Configure the screen and instantiate the faceplate type.
11. Display project in Runtime.

See also

Example: Creating faceplate types (Page 588)

Example: Configuring interface tags in faceplate types (Page 591)

Instead of tags: Using PLC user data type in the faceplate type (Page 594)

Example: Creating a local script for opening the pop-up (Page 600)

4.7.5.2 Example: Introduction

The example works with two motors and their data.

You configure a screen that displays data of the motor "Engine 01" in a faceplate instance.

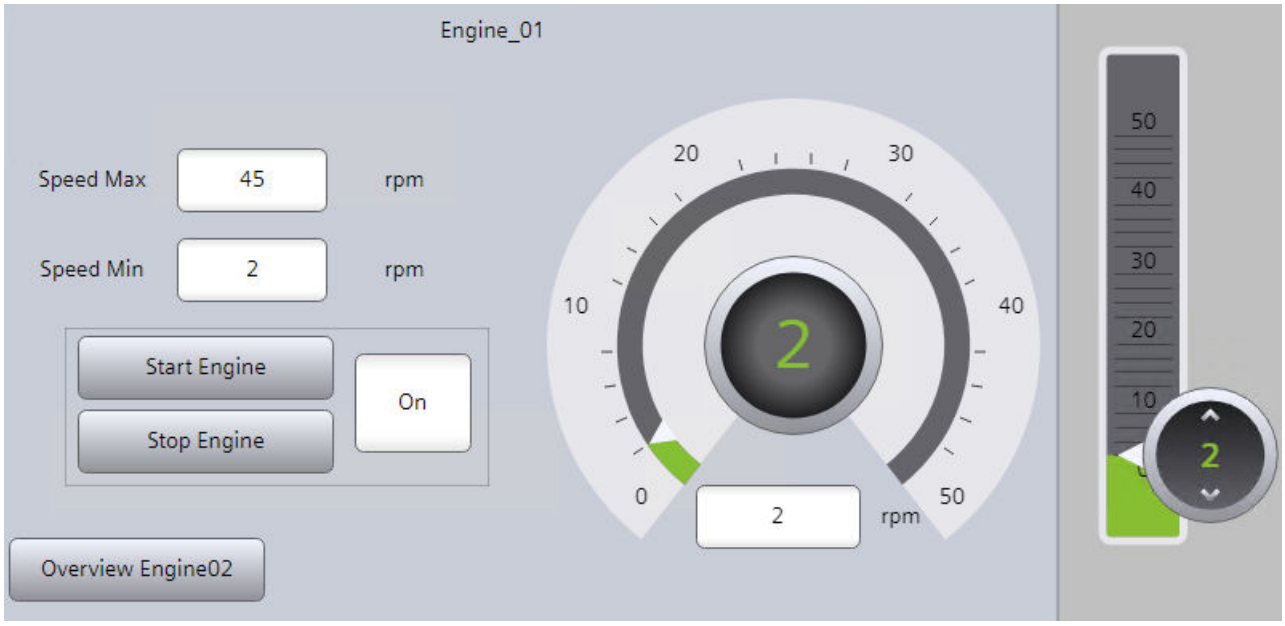
A second faceplate is instantiated in this faceplate, which has two buttons and a text box.

Press "Start Engine" and "Stop Engine" to change the speed of the motor to 2 or 0 rpm.

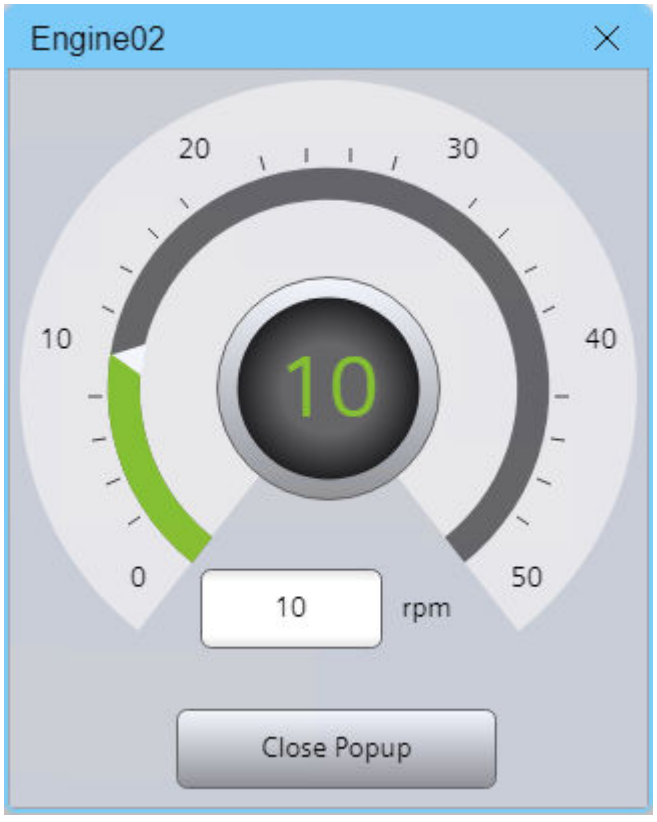
After pressing, the output field indicates whether the motor is in the status "On" or "Off".

Use the slider to adapt the speed.

If the speed exceeds a specified value, the output field under speed control changes its color from white to red.



By pressing another button within the faceplate type, you open a new faceplate type as a pop-up that displays the speed of the motor "Engine 02".



4.7.5.3 Example: Create HMI tags

Task

You create the HMI tags required for the project.

Requirement

- A project has been created.
- An HMI device has been configured.

Procedure

1. Open the "Standard tag table" editor under "HMI Tags" in the project tree.
2. Create the following tags in the "HMI tags" tab:

Tag name	Data type
Engine01_CurrentRPM	Int
Engine01_SpeedMax	Int
Engine01_SpeedMin	Int
Engine02_CurrentRPM	Int

3. Assign valid values to the tags in the Inspector window under "Properties > Properties > Values".

Result

You have configured the HMI tags required for the example.

4.7.5.4 Example: Creating faceplate types

Task

1. To create a faceplate type that is instantiated in the start screen. This faceplate type displays the "Engine 01" data in Runtime. Clicking a button opens another faceplate type as a popup.
2. You create a faceplate type that is opened as a pop-up from the faceplate type created in the first step when a button is clicked. This faceplate type displays the current speed of "Engine 02" in Runtime.
3. To create a faceplate type that is embedded in the faceplate from the first step and starts and stops the motor "Engine 01".

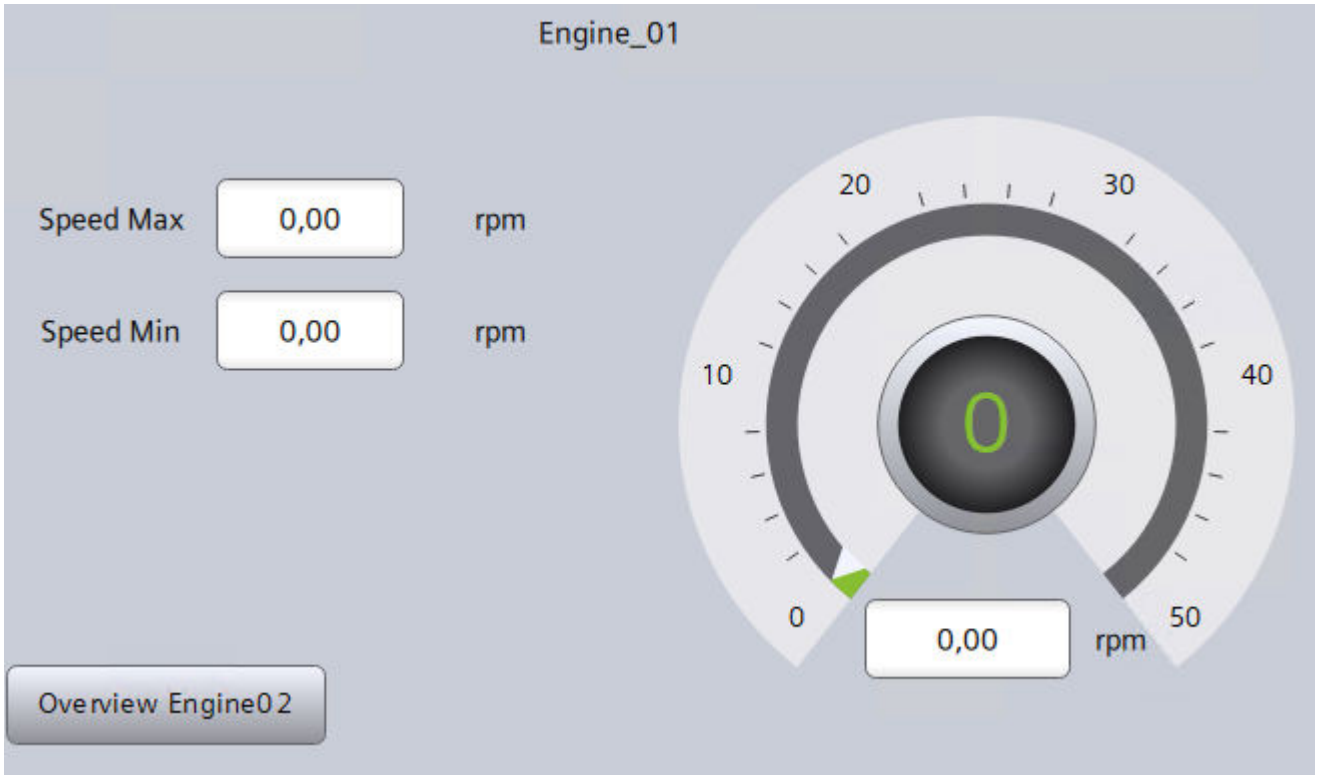
Requirement

- A project has been created.
- An HMI device has been configured.

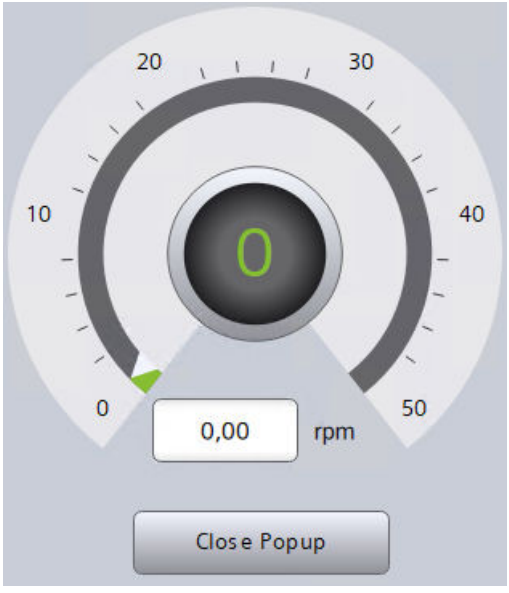
Procedure

1. To create the first faceplate for "Engine 01"", create a new faceplate type in the project library and name it accordingly, for example, "FP_Engine01".
   A version 0.0.1 of the faceplate type is being created and has the status "In progress".
   The version is opened in the "Visualization" tab.
2. Drag the required objects from the "Toolbox" task card to the faceplate type. The objects required for the example are listed and described in the table below.

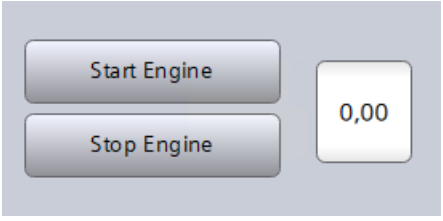
3. Arrange the objects according to your needs. This may look like this, for example:



4. To create the second faceplate type for "Engine 02", repeat steps 1 and 2. Name the faceplate type, for example, "FP_Engine02".
5. Arrange the objects according to your needs. This may look like this, for example:



6. To create the third faceplate type for "Engine StartStop", repeat steps 1 and 2. Name the faceplate type, for example, "FP_StartStop".
7. Arrange the objects according to your needs. This may look like this, for example:



Objects in the faceplate types

Faceplate type	Object	Object name	Meaning/Properties
FP_Engine01	Rectangle	Rectangle_1	Limitation and background of the display area
	Gauge	Gauge_1	Instrument display of the current motor speed
	Text box	Text box_1	Text "Engine_01"
	Text box	Text box_2	Text "Speed Max"
	Text box	Text box_3	Text "Speed Min"
	Text box	Text box_4	Text "rpm"
	Text box	Text box_5	Text "rpm"
	Text box	Text box_6	Text "rpm"
	IO field	IO field_1	Display of the HMI tag "Engine01_Speed-Min"
	IO field	IO field_2	Display of the HMI tag "Engine01_Speed-Max"
	IO field	IO field_3	Display of the current motor speed.
	Button	Button_1	Pressing the button opens a pop-up that shows the speed for Engine 02.
FP_Engine02	Rectangle	Rectangle_1	Limitation and background of the display area
	Gauge	Gauge_1	Instrument display of the current motor speed
	Text box	Text box_1	Text "rpm"
	IO field	IO field_1	Display of the current motor speed.
	Button	Button_1	Pressing the button closes the pop-up.
FP_StartStop	Rectangle	Rectangle_1	Limitation and background of the display area
	Button	Button_1	When the button is pressed, the speed of Engine 01 is set to the value 2.
	Button	Button_2	When the button is pressed, the speed of Engine 01 is set to the value 0.
	IO field	IO field_1	Display of the current motor status "On" or "Off".

Result

The required faceplate types have been created and are in the status "In progress". The objects required in the faceplate types are configured.

See also

Creating a faceplate type in the project library (Page 531)

Working with faceplate types and versions (Page 533)

4.7.5.5 Example: Configuring interface tags in faceplate types

Task

To configure tags in the faceplate type with which you can dynamically control properties and which are required for data exchange in the project.

Requirement

The faceplate types required for the example are created and are in the "In progress" state.

Configuring interface tags

Note

Interface tags are required for the exchange of values between the faceplate and the project.

When used in screens, access to individual object properties in faceplates is possible exclusively via faceplate tags.

When you follow the example with PLC user data types, use the PLC user data type accordingly. Note that the PLC user data type must exist in the project library in this case.

1. Open the faceplate type "FP_Engine01".
2. In the editor, switch to the "Tags Interface" tab.
3. Configure the required tags. To do this, click the button  or on "Add".
   The tags required for the various faceplate types are listed in the table below.
4. Repeat steps 1 to 3 for the faceplate types "FP_Engine02" and "FP_StartStop".

Interface tags of the faceplate types

Faceplate type	Tag	Data type	Source	Target
FP_Engine01	Engine01_CurrentRPM	Int	Current speed Engine 01 (Actual value)	IO field (Faceplate)
	Engine01_SpeedMax	Int	Specification of the maximum speed	IO field (Faceplate)
	Engine01_SpeedMin	Int	Specification of the minimum speed	IO field (Faceplate)
	Engine02_CurrentRPM	Int	Current speed Engine 02 (Actual value)	IO field (Faceplate)
FP_Engine02	Engine02_CurrentRPM	Int	Current speed Engine 02 (Actual value)	IO field (Faceplate)
FP_StartStop	Engine01_CurrentRPM	Int	Current speed Engine 01 (Actual value)	IO field (FP_Engine01)

Assigning interface tags in the faceplate types

To assign tags to the created objects in the faceplate types, follow these steps:

1. Open the Faceplate type in the "Visualization" tab.
2. Select the object to which you want to assign a tag.
3. Open the Inspector window under "Properties > Properties > General".
4. Under "Process value" in the "Dynamization" column, set the value "Tag".
   A screen for selecting the tag is shown on the right-hand side of the Inspector window.
5. Assign the corresponding tag to the "Process value" property. You will find the tag assignment in the table.
   If the tag is linked in the project, the object displays the corresponding values in Runtime.

Note

When you follow the example with PLC user data types, use the PLC user data type accordingly.

Tag assignment

Faceplate type	Object	Object name	Tag assignment
FP_Engine01	Gauge	Gauge_1	Engine01_CurrentRPM
	IO field	IO field_1	Engine01_SpeedMax
	IO field	IO field_2	Engine01_SpeedMin
	IO field	IO field_3	Engine02_CurrentRPM
FP_Engine02	Gauge	Gauge_1	Engine02_CurrentRPM
	IO field	IO field_1	Engine02_CurrentRPM
FP_StartStop	IO field	IO field_1	Engine01_CurrentRPM

See also

Instead of tags: Using PLC user data type in the faceplate type (Page 594)

Using a PLC user data type (Page 566)

4.7.5.6 Example: Configure local tags in faceplate types.

Task

You configure a local tag in the faceplate type "FP_StartStop". Clicking the "Button_1" or "Button_2" buttons changes the local tag to "On" or "Off". The I/O field "IO field_1" displays the value of the local tag.

Requirement

- The faceplate type "FP_StartStop" has been created.
- The faceplate type is in the "in progress" status.

Configuring the local tags

Note

Local tags are used to exchange values within the faceplate. When used in screens, access to individual object properties in faceplates is not possible via local tags.

1. Open the faceplate type "FP_StartStop".
2. In the editor, go to the "Local tags" tab.
3. Configure the tag "OnOff" as a WString. To do this, click the button  or on "Add".

Assigning local tags in the faceplate types

1. Open the "Visualization" tab.
2. Select the I/O field "IO field_1".
3. Open the Inspector window under "Properties > Properties > General".
4. Under "Process value" in the "Dynamization" column, set the value "Tag". A screen for selecting the tag is shown on the right-hand side of the Inspector window.
5. Assign the local tag "OnOff" to the "Process value" property. If the tag is linked in the project, the object displays the corresponding values in Runtime.

   Note

   When you follow the example with PLC user data types, use the PLC user data type accordingly.

Assigning values to local tag

To assign values to the configured local tag when the buttons are clicked, follow these steps:

1. Select the button "Button_1".
2. Open the Inspector window under "Properties > Events".
3. Select the event "Click left mouse button" and click the button.
   A new script is created.
4. Define the following script:
```
export function Button_Start_OnTapped(item, x, y, modifiers,
trigger){
  let tag1 = Tags("Engine01_CurrentRPM").Read();
  if(tag1 == 0){
    HMIRuntime.Tags.SysFct.SetTagValue("OnOff", "On");
  }
}
```
5. Repeat steps 2 and 3 for the "Button_2" button.
6. Select the event "Click left mouse button" and click the button.
   A new script is created.
7. Define the following script:
```
export function Button_Start_OnTapped(item, x, y, modifiers,
trigger){
  HMIRuntime.Tags.SysFct.SetTagValue("OnOff", "Off");
}
```

Note

The scripts are extended in the course of this example to meet the still missing requirements.

Result

You have configured a local tag and assigned it to an I/O field. When the buttons are clicked, a value is assigned to the local tag.

4.7.5.7 Instead of tags: Using PLC user data type in the faceplate type

Introduction

You have configured multiple motors of the same type. To do so, use data blocks based on a PLC user data type.

Task

You use the existing PLC user data type in the faceplate type.

You link the tags of the data blocks defined in the PLC user data type with those in the faceplate instance.

Requirement

- A SIMATIC S7-1200 or SIMATIC S7-1500 controller is configured.
- A PLC user data type suitable for the motor type is configured.
- At least one PLC tag based on the PLC user data type has been configured.
- The "Unified Faceplate Types" is opened with the created faceplate type.

Procedure

1. Drag the desired PLC user data type that you want to use in the project from the project tree into the project library under "Types".
2. Adapt the faceplate type so that the PLC user data type for the motors is also used in this faceplate type.
   - Switch to the "Tags interface" tab.
   - Select "PLCUDT" as the data type.
   - Under "User data type structure", select the defined PLC user data type.
     The PLC user data type supplies the faceplate type with the corresponding tags.
   - Switch back to the "Visualization" tab.
   - Keep in mind that tags are accessed in the faceplate type via the PLC user data type. This affects, for example, the use of faceplate tags in dynamized object properties and in local scripts.
3. Release the faceplate type.
4. From the faceplate type, create the same number of faceplate instances as the number of motors you have configured.
5. Link the tags from the data blocks and faceplate instances with each other.
   - The tag names in data blocks and faceplate instance are hierarchically structured.
   - This means that the tags from the individual faceplate instances can always be assigned uniquely to the tags of the respective data block.
   - When the PLC user data type contains tags that you do not require in the visualization, ignore these tags.

See also

Using a PLC user data type (Page 566)

4.7.5.8 Example: Instantiate the inner faceplate type in the outer faceplate type

Task

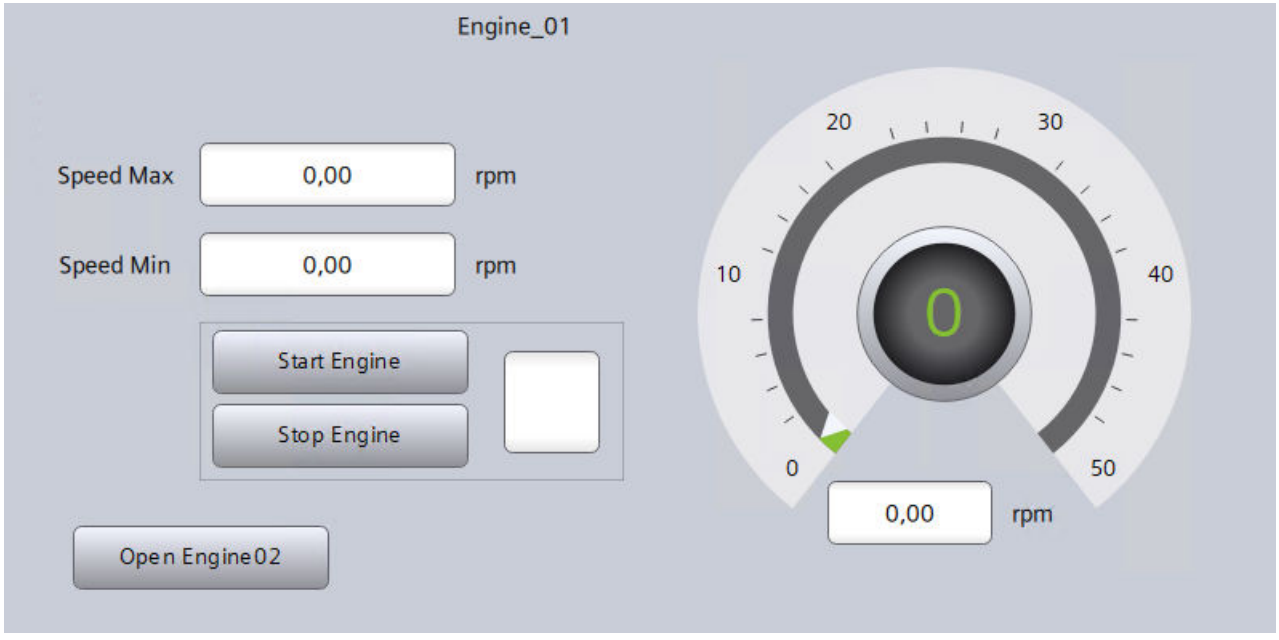
Insert the faceplate type "FP_StartStop" in the faceplate type "FP_Engine01".

Requirement

- The faceplate types "FP_StartStop" and "FP_Engine01" are created.
- The faceplate types are in the "In progress" status.

Procedure

1. Release the faceplate type "FP_StartStop".
2. Open the faceplate type "FP_Engine01" in the "Visualization" tab.
3. Drag-and-drop the "FP_Start_Stop" faceplate type into the "FP_Engine01" faceplate type at the desired location. This may look like this, for example:



Assigning interface tags to the inner faceplate type

To assign tags to the inner faceplate type "FP_StartStop" from the outer faceplate type "FP_Engine01", follow these steps:

1. Open the faceplate type "FP_Engine01" in the "Visualization" tab.
2. Select the faceplate instance from "FP_StartStop".
3. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface".
4. Assign the tag "Engine01_CurrentRPM" to the "Engine01_CurrentRPM" interface.

Result

An inner faceplate type is created in an outer faceplate type and the interface tags are assigned.

See also

Using a faceplate type in another faceplate type (Page 570)

Using a PLC user data type (Page 566)

Instead of tags: Using PLC user data type in the faceplate type (Page 594)

4.7.5.9 Example: Configuring interface properties in faceplate types

Task

Design the background color of the I/O field "IO field_3" (display box for the speed (actual value)) with interface properties. If a specified speed is exceeded, the background color of the I/O field should change from white to red.

Requirement

- The faceplate type "FP_Engine01" is configured.
- The faceplate type "FP_Engine01" is open for editing.

Configure interface property

1. Change to the "Property interface" tab in the editor.
2. Click the button  or on "Add".
3. Assign the name "BackgroundColor".
4. Select the "Color" data type:
5. Go to the "Visualization" tab.

Assign interface property

1. Select the I/O field "IO field_3".
2. Open the Inspector window under "Properties > Properties > Appearance".
3. Under "Background - color", in the "Dynamization" column, set the value to "Property interface".
   A screen for selecting the interface property is shown on the right side of the Inspector window.
4. Assign the interface property "BackgroundColor" to the "Background - color" property.

Result

You have configured an interface property that allows you to customize the background color in the faceplate instance.

The value of the interface property "BackgroundColor" adapts the background of the I/O field "IO field_3" in Runtime.

See also

Interface properties in faceplates (Page 548)

4.7.5.10 Example: Create an interface event

Task

You configure an interface event in the faceplate type "FP_StartStop". To set the speed to 0, use the interface event in the faceplate instance that is used in the faceplate type "FP_Engine01".

Requirement

- The faceplate type "FP_StartStop" is configured.
- The faceplate type "FP_StartStop" is open for editing.

Configuring an interface event

1. In the editor, go to the "Event interface" tab.
2. Click on the button or on the entry "Add".
   A new interface event is created.
3. Assign the name "Engine_Stop".
4. Click on the entry "Add" below the interface event.
   A new parameter is created. The data type of the parameter is "Int".
5. Assign the name "Stop".

Assigning an interface event

1. Select the "Button_2" button.
2. Open the Inspector window under "Properties > Events".
3. Select the event "Click left mouse button" and click on the button  .
4. Define the following script:

```
export function Button_Stop_OnTapped(item, x, y, modifiers,
trigger) {
   let parameters = {Stop:0};
   Faceplate.RaiseEvent("Stop_Engine", parameters);
   HMIRuntime.Tags.SysFct.SetTagValue("OnOff", "Off");
}
```

5. Release the faceplate type "FP_StartStop".

Calling an interface event in instance

1. Open the faceplate type "FP_Engine01".
2. Select the faceplate instance of the "FP_StartStop" faceplate type.

3. Open the Inspector window under "Properties > Events".
4. Select the "Stop_Engine" event and click the button .
5. Define the following script:

```
export function Faceplate_container_1_OnStop_Engine(item, Stop) {
   HMIRuntime.Tags.SysFct.SetTagValue("Engine01_CurrentRPM",
Stop);
}
```

Result

You have configured an interface event with which you can set the speed to the value 0.

4.7.5.11 Example: Using e script to change tags

Task

To increase the speed via the interface tag "Engine01_CurrentRPM", expand the script that runs when the "Button_1" button is clicked.

Requirement

- The faceplate type "FP_StartStop" is configured.
- The interface tag "Engine01_CurrentRPM" has been configured.
- The faceplate type "FP_StartStop" is open for editing.

Configuring a script

1. In the editor, switch to the "Visualization" tab.
2. Select the button "Button_1".
3. Open the Inspector window under "Properties > Events".
4. Select the event "Click left mouse button" and click on .
5. Define the following script:

```
export function Button_Start_OnTapped(item, x, y, modifiers,
trigger) {
   let tag1 = Tags("Engine01_CurrentRPM").Read();
   if (tag1 == 0) {
      HMIRuntime.Tags.SysFct.SetTagValue("Engine01_CurrentRPM", 2);
       HMIRuntime.Tags.SysFct.SetTagValue("OnOff", "On");
   }
}
```

Result

You have configured a script with which the speed is set to a value of 2 when you click the "Button_1" button.

4.7.5.12 Example: Creating a local script for opening the pop-up

Task

You create a script that runs when a button is clicked and transfers values to tags. Clicking the button in the faceplate type opens an additional faceplate type as a pop-up.

Requirement

- The faceplate types "FP_Engine01" and "FP_Engine02" are created.
- The "FP_Engine02" faceplate type is released in the version 0.0.1.

  Note

  If necessary, adapt the script below according to your released version.

  Replace spaces and periods with underscores.

- The faceplate type "FP_Engine01" is open for editing.

Procedure for creating the script: Opening the faceplate pop-up in the faceplate type

1. In the editor, switch to the "Visualization" tab.
2. Select the "Button_1" button.
3. Open the Inspector window under "Properties > Events".
4. Select the entry "Click left mouse button" and click on the button  .
5. Define the following script:

```
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
   let po = Faceplate.OpenFaceplateInPopup("FP_Engine02_V_0_0_1",
"Engine02", true, false);
   po.Left = 100;
   po.Top = 150;
   po.Visible = true;
}
```

   Note

   When you follow the example with PLC user data types, note the deviating tag names.

Result

You have configured a script at the button in the faceplate type that opens an additional faceplate as a pop-up.

See also

Dynamizing a faceplate type (Page 576)

Configure faceplate as pop-up (Page 581)

4.7.5.13 Example: Create local script to close the pop-up

Task

You create a script that runs when a button is clicked and closes the faceplate pop-up in which the button is located.

Requirement

- The faceplate types "FP_Engine01" and "FP_Engine02" are created.
- The faceplate type "FP_Engine02" is open for editing.

Procedure for creating the script: Closing the faceplate pop-up

1. In the editor, switch to the "Visualization" tab.
2. Select the button.
3. Open the Inspector window under "Properties > Events".
4. Select the entry "Click left mouse button" and click on the button  .
5. Define the following script:

```
export function Button_1_OnTapped(item, x, y, modifieres, trigger)
{
   Faceplate.Close();
}
```

Result

To close the faceplate pop-up, you have configured a script on the button in the faceplate type.

4.7.5.14 Example: Configure the screen and instantiate the faceplate type.

Task

You insert an instance of the faceplate type "FaceplateEngine01" in a screen and link the tags of the faceplate instance with tags in the project.

Requirement

- A screen has been created.
- The "Screens" editor is open.

Creating a faceplate instance

1. Release all created faceplate types.

   Note

   To avoid error messages about inconsistency, start with the inner faceplate when releasing the faceplates. In the example, this is "FP_StartStop". Then release the faceplate that will be instantiated in the script. In the example, this is "FP_Engine02". To finish, release "FP_Engine01".

2. Drag and drop the "FP_Engine01" faceplate type into the "Screens" editor.
   Alternatively, drag the "Faceplate container" object into the editor and link the container with the faceplate type.
3. Define the properties of the faceplate container.
4. Under "Interface", assign the appropriate tag to all existing interface tags.
   Assign a value for the "BackgroundColor" interface property.

Assigning tags

1. Select the faceplate instance or the faceplate container.
2. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface".
3. Assign the appropriate HMI tag to all existing interface tags.

Assigning interface properties

1. Select the faceplate instance or the faceplate container.
2. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface".
3. Under "BackgroundColor", in the "Dynamization" column, set the value to "Script".
4. Create the following script:

```
export function
Faceplate_Container_1_Properties_BackgroundColor_Trigger (item) {
   var value;
   let ProcessValue = Tags("Engine01_CurrentRPM").Read();
   if (ProcessValue > 40) {
      value = HMIRuntime.Math.RGB(255,0,0)
   }
   else {
      value = HMIRuntime.Math.RGB(255,255,255)
   }
   return value;
}
```

The background color of the I/O field "IO field_3" in faceplate type "FP_Engine01" is white up to a speed of 40. When this speed is exceeded, the background color changes to red.

Inserting a slider

1. Drag and drop a slider from the "Toolbox" task card into the screen.
2. Select the slider and open the Inspector window under "Properties > Properties > General".
3. Assign the "Engine01_CurrentRPM" tag to the "Process value" property.

Result

The faceplate type is instantiated on a screen in a faceplate container.

You have linked the interface tags with project tags.

You have assigned a dynamic value to the "BackgroundColor" property of the I/O field "IO field_3".

You have configured a slider that can be used to adapt the speed "Engine01_CurrentRPM" in Runtime.

This completes the integration of the faceplate type into the project.

See also

Creating a faceplate instance (Page 564)

4.7.5.15 Example: Displaying a project in runtime.

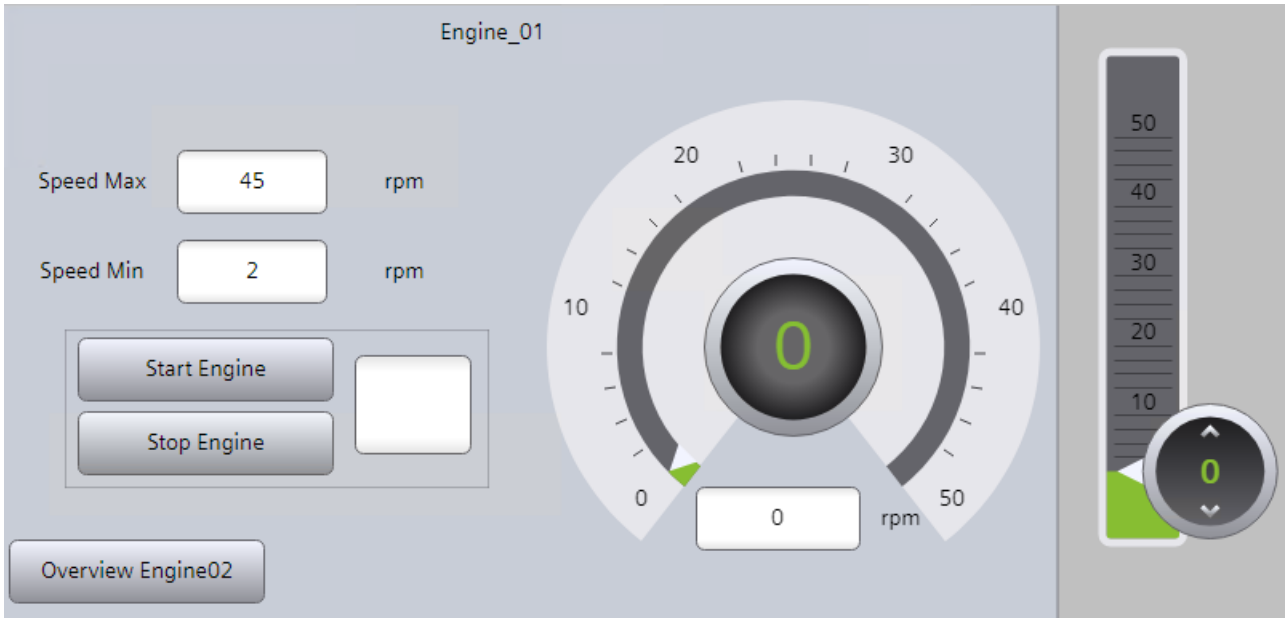
Procedure

1. Compile and upload the project to the HMI device.
2. Open the Runtime on the HMI device.

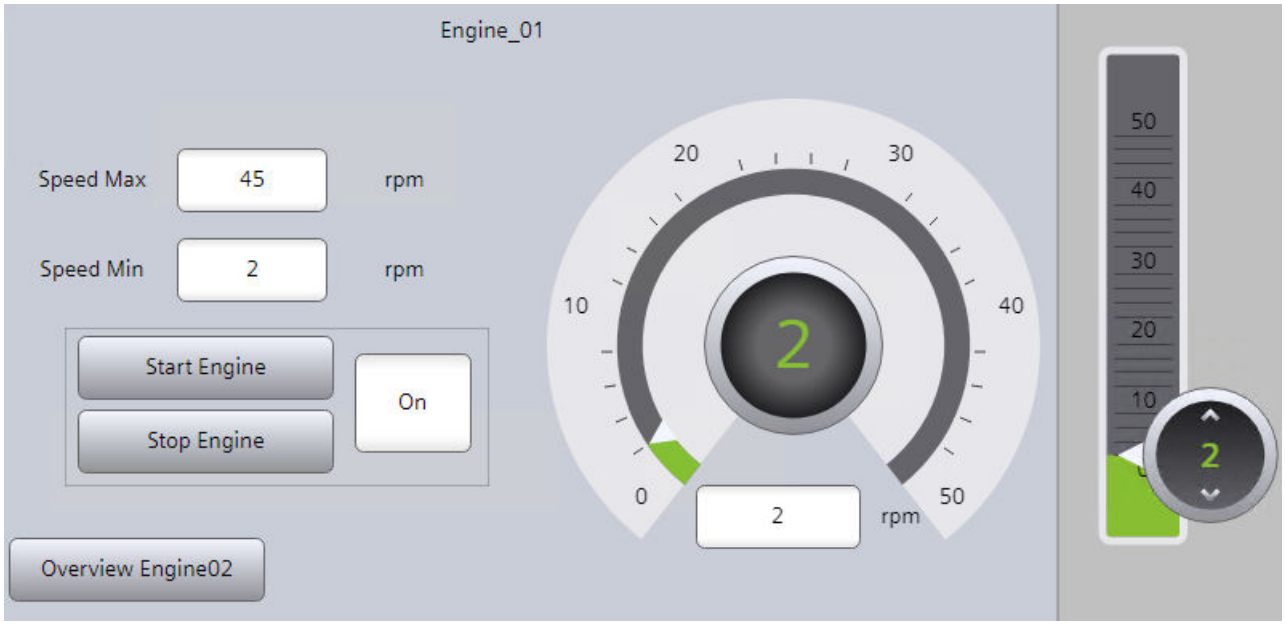
Result

When you have implemented the project as shown in the example, you will see the following elements in Runtime that display the values you have specified:

1. Start screen:



2. After pressing the button to start the motor:



3. After pressing the button to open the pop-up:


Engine_01
Speed Max
45
rpm
Speed
Overvi
Engine02
10
rpm
Close Popup
2
rpm
2
50
40
30
20
10
2

Configuring tags

5

5.1 Basics

5.1.1 Basics of tags

Introduction

Process values are forwarded in runtime using tags. Process values are data which is stored in the memory of one of the connected automation systems. They represent the status of a plant in the form of temperatures, fill levels or switching states, for example. Define external tags for processing the process values in WinCC.

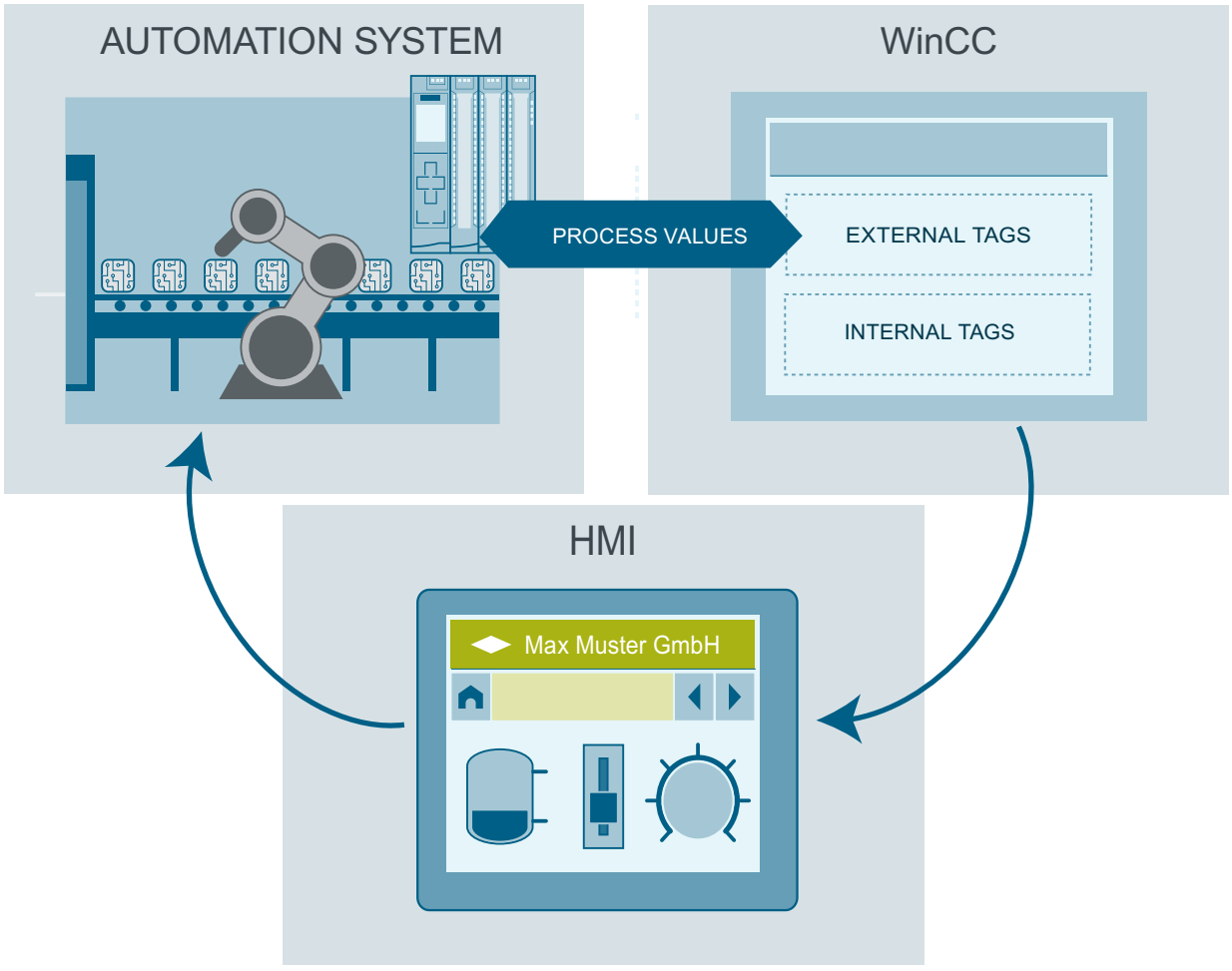
WinCC works with two types of tag:

- External tags
- Internal tags

The external tags form the link between WinCC and the automation systems. The values of external tags correspond to the process values from the memory of an automation system. The value of an external tag is determined by reading the process value from the memory of the automation system. It is also possible to rewrite a process value in the memory of the automation system.

Internal tags do not have a process link and only convey values within the WinCC. The tag values are only available as long as runtime is running.

In WinCC, you can visualize and change process values, which are transferred using tags, on your HMI device.



Tags in WinCC

For external tags, the properties of the tag are used to define the connection that the WinCC uses to communicate with the automation system and form of data exchange.

Tags that are not supplied with values by the process - the internal tags - are not connected to the automation system. In the tag's "Connection" property, this is identified by the "Internal tag" entry.

You can create tags in different tag tables for greater clarity. You then directly access the individual tag tables in the "HMI tags" node in the project tree. The tags from all tag tables can be displayed with the help of the table "Show all tags".

With structures you bundle a number of different tags that form one logical unit. Structures are project-associated data and are available for all HMI devices of the project. You use the "Types" editor in the project library to create and edit a structure.

See also

Overview of HMI tag tables (Page 609)

External tags (Page 610)

Internal tags (Page 615)

5.1.2 Overview of HMI tag tables

Introduction

HMI tag tables contain the definitions of the HMI tags that apply across all devices. A tag table is created automatically for each HMI device created in the project.

In the project tree there is an "HMI tags" folder for each HMI device. The following tables can be contained in this folder:

- Default tag table
- User-defined tag tables
- Table of all tags

In the project tree you can create additional tag tables in the "HMI tags" folder and use these to sort and group tags and constants. You can move tags to a different tag table using a drag-and-drop operation or with the help of the "Tag table" field. Activate the "Tags table" field using the shortcut menu of the column headings.

In WinCC you can display the locations of use for all tags. Use the "Cross-references" command in the shortcut menu or the F11 key to call the "Cross-references" editor for a selected tag table. In the editor you can see all objects that the respective tag uses and you can jump directly to the location of use of the tag.

Default tag table

There is one default tag table for each HMI device of the project. It cannot be deleted or moved. The default tag table contains HMI tags and, depending on the HMI device, also system tags. You can declare all HMI tags in the standard tag table or, as necessary, in additional user-defined tag tables.

User-defined tag tables

You can create multiple user-defined tag tables for each HMI device in order to group tags according to your requirements. You can rename, gather into groups, or delete user-defined tag tables. To group tag tables, create additional subfolders in the HMI tags folder.

Show all tags

The "HMI tags" tab in the "All tags" table shows an overview of all HMI tags and system tags of the HMI device in question. This table cannot be deleted, renamed or moved. The "Tags table" column shows you in which tag table a tag is included. Using the "Tags table" field, the assignment of a tag to a tags table can be changed.

The "All tags" table contains an additional tab "System tags". The system tags are created by the system and used for internal management of the project. The names of the system tags begin with the "@" character. System tags cannot be deleted or renamed. You can evaluate the value of a system tag, but cannot modify it.

Discrete alarms, analog alarms and logging tags

The following tables are also available in an HMI tag table:

- Discrete alarms
  In the "Discrete alarms" table, you configure discrete alarms to the HMI tag selected in the HMI tag table. When you configure a discrete alarm, multiple selection in the HMI tag table is not possible. You configure the discrete alarms for each HMI tag separately.
- Analog alarms
  In the "Analog alarms" table, you configure analog alarms to the HMI tag selected in the HMI tag table. When you configure an analog alarm, multiple selection in the HMI tag table is not possible. You configure the analog alarms for each HMI tag separately.
- Logging tags
  In the "Logging tags" table, you configure logging tags to the HMI tag selected in the HMI tag table. When you configure a logging tag, multiple selection in the HMI tag table is not possible. You configure the logging tags for each HMI tag separately.

With the help of these tables you configure alarms and logging tags for the currently selected HMI tag.

5.1.3 External tags

Introduction

External tags allow the data exchange between the components of an automation system, for example, between an HMI device and a PLC.

An external tag is the image of a defined memory location in the PLC. You have read and write access to this storage location from both the HMI device and from the PLC.

As external tags map a storage location in the PLC, the applicable data types depend on the PLC that is connected to the HMI device.

If you write a PLC control program in STEP 7, the PLC tags created in the control program will be added to the PLC tag table. If you want to connect an external tag to a PLC tag, access the PLC tags directly via the PLC tag table and connect them to the external tag.

Data types

All simple data types available on the connected PLC are available at an external tag in WinCC. If the data type of the PLC tag is not available in WinCC, a compatible data type is automatically used at the HMI tag. Interconnected PLC data types and arrays are not supported.

If you use PLC data types, the data type is adopted by WinCC. You can change the data type at the HMI tag, if necessary.

Central tag management in STEP 7

You can connect also connect DB instances of user-defined PLC data types (UDT) to the HMI tags.

The PLC data type and the corresponding DB instances are created and updated centrally in STEP 7. In WinCC, you can use the following sources as the PLC tag (DB instances):

- Data block elements that use a UDT as data type
- Data block instances of a UDT

The data type is taken from STEP 7 and is not converted to an HMI data type. The access type is always "Symbolic access". Depending on the release for WinCC in STEP 7, elements and structured elements of the PLC data type are applied to WinCC.

Note

Accessing PLC data types

Access to PLC data types is only available in conjunction with SIMATIC S7-1500.

Synchronization with PLC tags

A variety of options for synchronizing external tags with the PLC tags are available in the runtime settings under "Settings for tags".

When you perform synchronization, you have the option of automatically applying the tag names of the PLC to external tags and reconnecting the existing tags.

The generated tag name is derived from the position of the data value in the hierarchical structure of the data block.

Update of tag values

For external tags, the current tag values are transmitted in runtime via the communication connection between WinCC and the connected automation systems and then saved in the runtime memory. Next, the tag value will be updated to the set cycle time. For use in the runtime project, WinCC accesses tag values in the runtime memory that were read from the PLC at the previous cycle time. As a result, the value in the PLC can already change while the value from the runtime memory is being processed.

See also

Creating external tags (Page 626)

5.1.4 Addressing external tags

Introduction

The options for addressing external tags depend on the type of connection between WinCC and the PLC in question. A distinction must be made between the following connection types:

- Integrated connection
  Connections of devices which are within a project and were created with the "Devices & Networks" editor are referred to as integrated connections.
- Non-integrated connection
  Connections of devices which were created with the "Connections" editor are referred to as non-integrated connections. It is not necessary that all of the devices be within a single project.

The connection type can also be recognized by its icon.

	Integrated connection
	Non-integrated connection

Addressing with integrated connections

An integrated connection offers the advantage that you can address a tag both symbolically and absolutely.

For symbolic addressing, you select the PLC tag via its name and connect it to the HMI tag. The valid data type for the HMI tag is automatically selected by the system.

During the symbolic addressing of a data block with optimized access and standard access, the address of an element in the data block is dynamically assigned and is automatically adopted in the HMI tag in the event of a change. You do not need to compile the connected data block or the WinCC project for this step.
For data blocks with optimized access, only symbolic addressing is available.

For symbolic addressing of elements in a data block, you only need to recompile and reload the WinCC project in case of the following changes:

- If the name or the data type of the linked data block element or global PLC tag has changed.
- If the name or the data type of the higher level structure node of a linked element in the data block element or global PLC tag has changed.
- If the name of the connected data block has changed.

Symbolic addressing is currently available with the following PLCs:

- SIMATIC S7-1500
- SIMATIC ET 200 CPU
- SIMATIC S7-1500 software controller

Symbolic addressing is also available if you have an integrated link.

You can also use absolute addressing with an integrated connection. You have to use absolute addressing for PLC tags from a SIMATIC S7-300/400 PLC. If you have connected an HMI tag with a PLC tag and the address of the PLC tag changes, you only have to recompile

the control program to update the new address in WinCC. Then you recompile the WinCC project and load it onto the HMI device.

In WinCC, symbolic addressing is the default method. To change the default setting, select the menu command "Options > Settings > Visualization > HMI tags".

The availability of an integrated connection depends on the PLC used. The following table shows the availability:

Controller	Integrated connection	Comments
S7-300/400	Yes	The linking of tags is not checked in runtime. If the tag address changes in the PLC and the HMI device is not compiled again and loaded, the change is not registered in runtime.
S7-1500	Yes	A validity check of the tag connection is performed in runtime during symbolic addressing. If an address is changed in the PLC, the change is registered and an error message is issued. In the case of absolute addressing, the behavior described for the S7-300/400 applies.
SIMATIC ET 200 CPU	Yes	A validity check of the tag connection is performed in runtime during symbolic addressing. If an address is changed in the PLC, the change is registered and an error message is issued. In the case of absolute addressing, the behavior described for the S7-300/400 applies.
SIMATIC S7-1500 software controller	Yes	A validity check of the tag connection is performed in runtime during symbolic addressing. If an address is changed in the PLC, the change is registered and an error message is issued. In the case of absolute addressing, the behavior described for the S7-300/400 applies.

Create an integrated connection in the "Devices & Networks" editor. If the PLC is contained in the project and integrated connections are supported, you can then also have the connection created automatically. To do this, when configuring the HMI tag, simply select an existing PLC tag to which you want to connect the HMI tag. The integrated connection is then automatically created by the system.

Addressing with non-integrated connections

In the case of a project with a non-integrated connection, you always configure a tag connection with absolute addressing. Select the valid data type yourself. If the address of a PLC tag changes in a project with a non-integrated connection during the course of the project, you also have to make the change in WinCC. The tag connection cannot be checked for validity in runtime, an error message is not issued.

A non-integrated connection is available for all supported PLCs.

Symbolic addressing is not available in a non-integrated connection.

With a non-integrated connection, the control program does not need to be part of the WinCC project. You can perform the configuration of the PLC and the WinCC project independently of each other. For configuration in WinCC, only the addresses used in the PLC and their function have to be known.

5.1.5 Indirect addressing

Absolute address multiplexing

For Unified Runtime PCs and Panels, absolute address multiplexing was enabled for HMI tags with S7 1200/1500 and S7 300/400 connections. You can configure the tags as placeholders for the address in the S7 1200/1500 and S7 300/400 PLCs. With a single tag you access a large number of memory locations in the address range of the PLC.

Product requirements

- You determine an absolute multiplex address via:
  - Address dialog
  - Text input
  - Copy/paste string
  - Openness
  - Excel Export/Import
- Address multiplexing is provided for integrated and non-integrated connections.
- You can use any internal or external tag with a numeric data type from the same device as the index tag.
- Address multiplexing is possible for all elementary data types.
- There is no support for the arrays on S7 1200/1500 and S7 300/400.
- Address multiplexing must not be provided for any structured data type.
- When using the global search, you can find the corresponding multiplex address and jump to it.
- You get a replacement text in the PLC tag property if the tag is used as multiplexing.
- You can work with multiplexed tags in the runtime.

Design requirements

- Implement tag multiplexing based on the classic HMI tag for the Unified devices.
- Consider and support existing NFR aspects of the HMI tag, specifically performance, delta compile, quantity structures, user friendliness, and testability.

5.1.6 Internal tags

Introduction

Internal tags do not have any connection to the PLC.

Principle

Internal tags are stored in the memory of the HMI device. Therefore, only this HMI device has read and write access to the internal tags. You can create internal tags to perform local calculations, for example.

You can use the HMI data types for internal tags. Availability depends on the HMI device being used.

The following HMI data types are available:

HMI data type	Data format
Bool	Binary tag
Byte	Unsigned 8-bit value
DateTime	Date/time format
DInt	Signed 32-bit value
DWord	Unsigned 32-bit value
Int	Signed 16-bit value
LInt	Signed 32-bit value
LReal	Floating-point number 64-bit IEEE 754
LTime	Signed duration
LWord	Unsigned 64-bit value
Real	Floating-point number 32-bit IEEE 754
SInt	Signed 8-bit value
UDInt	Unsigned 32-bit value
UInt	Unsigned 16-bit value
ULInt	Unsigned 64-bit value
USInt	Unsigned 8-bit value
WChar	Text tag, 16-bit character set
Word	Unsigned 16-bit value
WString	Text tag, 16-bit character set
Array	Data structure

See also

System tags (Page 616)

5.1.7 System tags

Introduction

System tags are internal tags that are required for internal management of the project.

Principle

The system tag names always start with the "@" character. You may not delete or rename these tags. You cannot change the value of a tag.

Note

You must not create any tags whose name starts with a @.

By default, the following system tags are available for the WinCC Unified devices:

System tag	Data type	Meaning
@CurrentLanguage	UDInt	Contains the current runtime language.
@DeltaActivationState	UDInt	Contains the current phase when loading changes.
@DiagnosticsIndicatorTag	UDInt	Specifies the diagnostic status. The diagnostic status contains the collective status of all relevant PLCs. The worst status of all PLCs is always used in this case. "@DiagnosticsIndicatorTag" can have the following values: • Uncertain (0) • Good (1) • Maintenance (2) • Error (3)
@LocalMachineName	WString	Contains the name of the local computer. "@LocalMachineName" is a local system tag of the current session. This means the system tag cannot be used as the trigger of a Scheduled task, for example.
@ServerMachineName	WString	Contains the name of the PC on which Runtime is running.
@SystemActivationState	UDInt	Signals whether Runtime is active and can have the following values: • System startup in progress (1) • System started (activated) (2) • System stopped (3) • System shutdown in progress (4) • System restart in progress (5)

System tag	Data type	Meaning
@SystemHealthIndex	UDInt	Shows the current status of the Runtime system. When the "@SystemHealthIndex" value is 0, the Runtime system is in optimal health. Runtime is in the state expected by the Engineering System. There are no messages that may have a negative impact on the "@SystemHealthIndex". The greater the "@SystemHealthIndex" value, the less healthy the state of the Runtime system.
@UserName	WString	Contains the currently logged-on user. "@UserName" is a local system tag of the current session. This means the system tag cannot be used as the trigger of a Scheduled task, for example.

See also

Internal tags (Page 615)

Shutdown behavior (Page 7827)

5.1.8 Updating the tag value in runtime

Introduction

Tags contain process values which change while runtime is running. Value changes are handled differently at internal and external tags.

Principle

When runtime starts, the value of a tag is equal to its start value. Tag values change in runtime.

In runtime, you have the following options for changing the value of a tag:

- A value change in an external tag in the PLC.
- By input, for example, in an I/O field.
- A value assignment in a script.

Updating the value of external tags

The value of an external tag is updated as follows:

- Cyclic in operation
  If you select the "Cyclic in operation" acquisition mode, the tag is updated in runtime while it is displayed in a screen or is logged. The acquisition cycle determines the update cycle for tag value updates on the HMI device. You can either choose a default acquisition cycle or define a user-specific cycle.
- On demand
  If you select the "On demand" acquisition mode, the tag is not updated cyclically. It will only be updated on demand using the "UpdateTag" system function, for example, or by a script.

See also

Defining the acquisition cycle for a tag (Page 635)

5.1.9 Limits and start values of a tag

Introduction

You can configure start values and restrict the value ranges with limits for numerical tags.

Use the limits to warn the operator when the value of a tag enters a critical range, for example.

Use the start values to assign a default value to an I/O field that is specified as start value in the linked tag.

Tags limits

You can specify a value range defined by a high limit and a low limit for numerical tags.

You configure four limit values that limit the value range. Using the limits Upper 2 and Lower 2 , you specify the maximum and minimum value for the value range. The limits Upper 1 and Lower 1 specify the threshold values at which the normal range is exceeded or undershot.

Limit	Application
Upper 2	Specifies the maximum value.
Lower 2	Specifies the minimum value.

If the operator enters a value for the tag that is outside the configured value range, the input is rejected. When the tag value leaves the value range, the function list is processed.

Note

If you want to output an analog alarm when a limit is violated, configure the respective tag in the "Analog alarms" tab. You can also configure the analog alarm in the "HMI alarms" editor. The values for output of an analog alarm depend on the configured tag limits.

Note

Limitation

Tags with the following data types do not support limits:

- Array and array elements
- Byte
- Char
- DWord, LWord and Word

Start value of a tag

You can configure a start value for numeric tags and tags for date/time values. The tag will be preset to this value when runtime starts. In this way, you can ensure that the tag has a defined status when runtime starts.

The start value cannot have the data type Raw or TextRef.

For external tags, the start value will be displayed on the HMI device until it is overwritten by the PLC or by input.

If no start value was configured, the tag contains the value "0" when starting runtime.

Use the "Persistence" setting to specify whether the value of the tag is to be retained when runtime is closed. The value saved will be used as the start value when you restart runtime.

See also

Defining limits for a tag (Page 637)

5.1.10 Data logging

Introduction

Data logging is used to collect, process and log process data from an industrial system. When you analyze the logged process data, you can extract important business and technical information regarding the operational state of the system.

The process values to be logged are compiled, processed and saved in the log database in runtime. Current or previously logged process values can be output in runtime as a table or trend. In addition, it is possible to print out logged process values as a report.

Use

You can use data logging for the following tasks:

- Early detection of danger / fault states
- Increase of productivity
- Enhancement of product quality

- Optimization of maintenance cycles
- Documentation of process value trends

Configuration

You configure the logging of process values in the "Logs" editor. You create a data log and an alarm log. The data log stores process values in logging tags. When you configure the data log, select the storage location, the logging period and the size of the log. You also specify the settings for the logging segments.

You configure trend views and process controls for displaying process data in runtime in the "Screens" editor. These views allow you to output the process data in the form of trends and tables.

Logging tags

You can log the values of internal and external tags. Use the logging tags for logging tag values. In logging tags, you specify how the values of the corresponding tags are written to the log.

You can create a logging tag for each HMI tag in the tag tables. You define the logging tags in the "HMI Tags" editor under "Logging tags". You specify for each logging tag to which log the tag is written.

With the default logging type "On change", the process value is compared to the saved value and the new value is only written to the log if the process value has changed.

To preserve memory, you can activate "smoothing" for the logging tags. By doing so, insignificantly small changes are filtered out prior to writing and the number of logged values is reduced.

Another way to save storage space is to use "Compression" for the logging tags. You select one of the 8 compression modes that calculate the values to be stored.

You create logging tags for internal and external tags. When logging the PLC tag values, the time stamp contains the time at which the value occurred in the PLC.

Note

Supported data type for logging external tags

When logging external PLC tags, all data types are supported except "Raw", "TextRef", "Struct" and "Array". The data type "TextRef" must not be used within the "Faceplate container" object.

5.1.11 Basics of tag management

Basics of tag management

You can always rename, copy or delete tags.

When a tag is renamed, the new name must be unique for the whole device.

Note

The connection to the tags can be interrupted in runtime under the following conditions during renaming:

- HMI tag is used in a type, for example, to dynamize an object property via a script.
- One or more instances of the type are used in the project.
- Project is in runtime.
- After the renaming, execute the command "Compile > Software (only changes)".

Solution: Exit runtime and rename the tag. Execute the "Compile > Software (rebuild all)" command.

If you use the "Copy" command to copy a tag to the clipboard, the references are copied along with the tag.

If you use the "Insert" command to add a tag to another device, the tag will be added without the connected references. Only the object name of the reference will be inserted. If a reference of the same name and valid properties exists in the target system, the existing reference will then be connected to the copied tag.

If you copy a tag, some of the objects linked to the tag are copied as well. The following objects are copied:

- Logging tags
- Cycles
- Alarms

If you add the copied tag to another device, the tag is added together with the linked objects.

Before you delete a tag, check in the "Cross-references" editor where the tag is used and what impact the deletion of the tag will have on your project.

5.1.12 Basics of user data types

Introduction

With user data types you bundle a number of different tags that form one logical unit. You create a user data type as a type and use instances of this type in the project. User data types are project-associated data and are available for all HMI devices of the project.

WinCC also supports the connection of PLC data types (UDTs) as user data types.

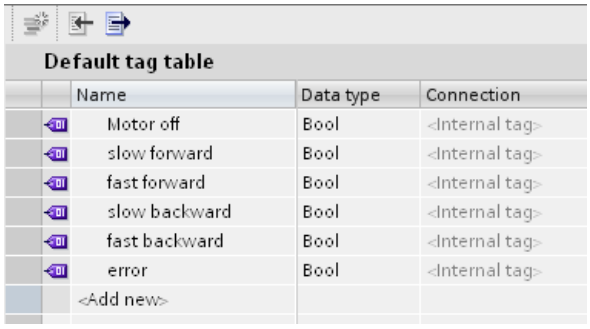
User data types also differ in their applicability with a specific communication driver. User data types are available for the following communication drivers:

- SIMATIC S7-300/400
- SIMATIC S7-1500

Create user data types and user data type elements in the project library.

Principle

For example, the different conditions of a motor can be described using 6 unique Boolean tags.



Default tag table

	Name	Data type	Connection
	Motor off	Bool	<Internal tag>
	slow forward	Bool	<Internal tag>
	fast forward	Bool	<Internal tag>
	slow backward	Bool	<Internal tag>
	fast backward	Bool	<Internal tag>
	error	Bool	<Internal tag>
	<Add new>		

If the overall condition should be described with a single tag, this tag can be created based on a user data type. For each of the individual Boolean tags you create a user data type element in the user data type.

This user data type can then be assigned complete to a faceplate for the motor. The created and released version of user data type is displayed at the tag in the "Data type" selection field.

The configured properties of a user data type are used in the instances of the user data type. If required, you change individual properties directly at the point of use, e.g. at a tag. Changing a property at the tag does not affect the user data type created.

5.1.13 Export and import of tags

Introduction

With Export and Import, you have the option to export tags from one project and import them into another project. There is also the option to create larger numbers of tags outside of WinCC, edit them and subsequently import into any WinCC project.

You export and import tags with the "Import" and "Export" buttons in the "HMI Tags" editor. When importing the tag data to WinCC, pay attention to the structure required in the import file.

Tag data structure

The tag data file must be in "*.xlsx" data format for the tag import and must be structured according to specific rules.

The import file in Microsoft Excel consists of a number of worksheets:

- HMI Tags (tags)
- SubstituteValueUsage (substitute value)

Each tag is on a separate row in the import file. The import file with the tag data must have the following format:

Example of the worksheet "HMI Tags"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Name	Path	Connection	PLC tag	DataType	HMI DataTy	Length	Address	Access Me	Start valu	Quality Co	Persisten	Substitute	Tag value	Update M	Coding	Comment	High High
2	Tag_1	Default table	HMI_Connection_1	Data_block_1.Static_1	Real	Float	4	%DB1.DBD(	<No Value	<No Value	True	False	25	<No Value	Client/Ser	Binary	<No Value	NoLimit
3	Tag_2	Default table	<No Value>	<No Value>	Int	Int	2	<No Value>	<No Value	<No Value	False	False	<No Value	<No Value	Client/Ser	Binary	<No Value	NoLimit

Table 5-1 Meaning of the entries

List entry	Meaning
Name	Indicates the configured name of an HMI tag.
Path	Specifies which folders in the project tree contain the tag. The folder structure is represented by ",": "FolderName1,FolderName2,TagName".
PLC Tag	Specifies which PLC tag is linked to the HMI tag.
Connection	Indicates the name of the connection.
Date type	Specifies the data type of the PLC tag. The data types allowed depend on the communication driver being used. See the "Communication" section of the documentation for additional information on the data types permitted for the various communication drivers.
HMI Data type	Specifies the data type of the HMI tag. The data types allowed depend on the communication driver being used. See the "Communication" section of the documentation for additional information on the data types permitted for the various communication drivers.
Length	Specifies the length of the tag in bytes. This entry is only useful for data types with a dynamic length, for example, strings. This entry remains empty for other data types.
Access Method	Specifies the type of access.
Address	Specifies the tag address in the PLC. The tag address must exactly match the one used in WinCC, for example, "%DB1.DBW0". The tag address is empty for internal tags.
Start Value	Specifies the start value of a tag.
Quality code	Provides information on the quality of the connection.
Persistency	Indicates whether the value is to be retained after the end of runtime.
Substitute Value	Indicates the substitute value. The substitute value is used when a process value with errors is read.
ID tag	The update ID updates the value of a tag with the aid of a function or a PLC job. The update ID must be unique within an HMI device.
Update Mode	Indicates whether the tag is to be updated locally or for the entire project.
Acquisition mode	Indicates the acquisition mode.
Acquisition cycle	Indicates the acquisition cycle used.
Limit Upper 2 type	Indicates whether the limit value "Upper 2" is monitored by a constant or not at all.
Limit Upper 2	Displays the limit value "Upper 2".
Limit Lower 2 type	Indicates whether the limit value "Lower 2" is monitored by a constant or not at all.
Limit Lower 2	Displays the limit value "Lower 2".
End value PLC	Specifies the end value of the PLC tag.
Start value PLC	Specifies the start value of the PLC tag.
End value HMI	Specifies the end value of the HMI tag.
Start value HMI	Specifies the start value of the HMI tag.

Example of the worksheet "SubstituteValueUsage"

	A	B
1	HMI Tag name	Substitute Value Usage
2	Tag_1	StartValue
3	Tag_1	ConnectionError

Table 5-2 Meaning of the entries

List entry	Meaning
HMI Tag name	Specifies the tag for which a substitute value has been defined. The tag must be available in the "HMI-Tags" worksheet.
Substitute Value Usage	Indicates the substitute value. The substitute value can be used in the following situations: • As start value • After communication error • Upper 2 limit value • Lower 2 limit value

Note

"No value" in the table

Entries in the table which have the value "No value" delete the corresponding values in an existing tag of the same name.

See also

Importing and exporting tags (Page 641)

5.2 Configuring tags

5.2.1 Working with tag tables

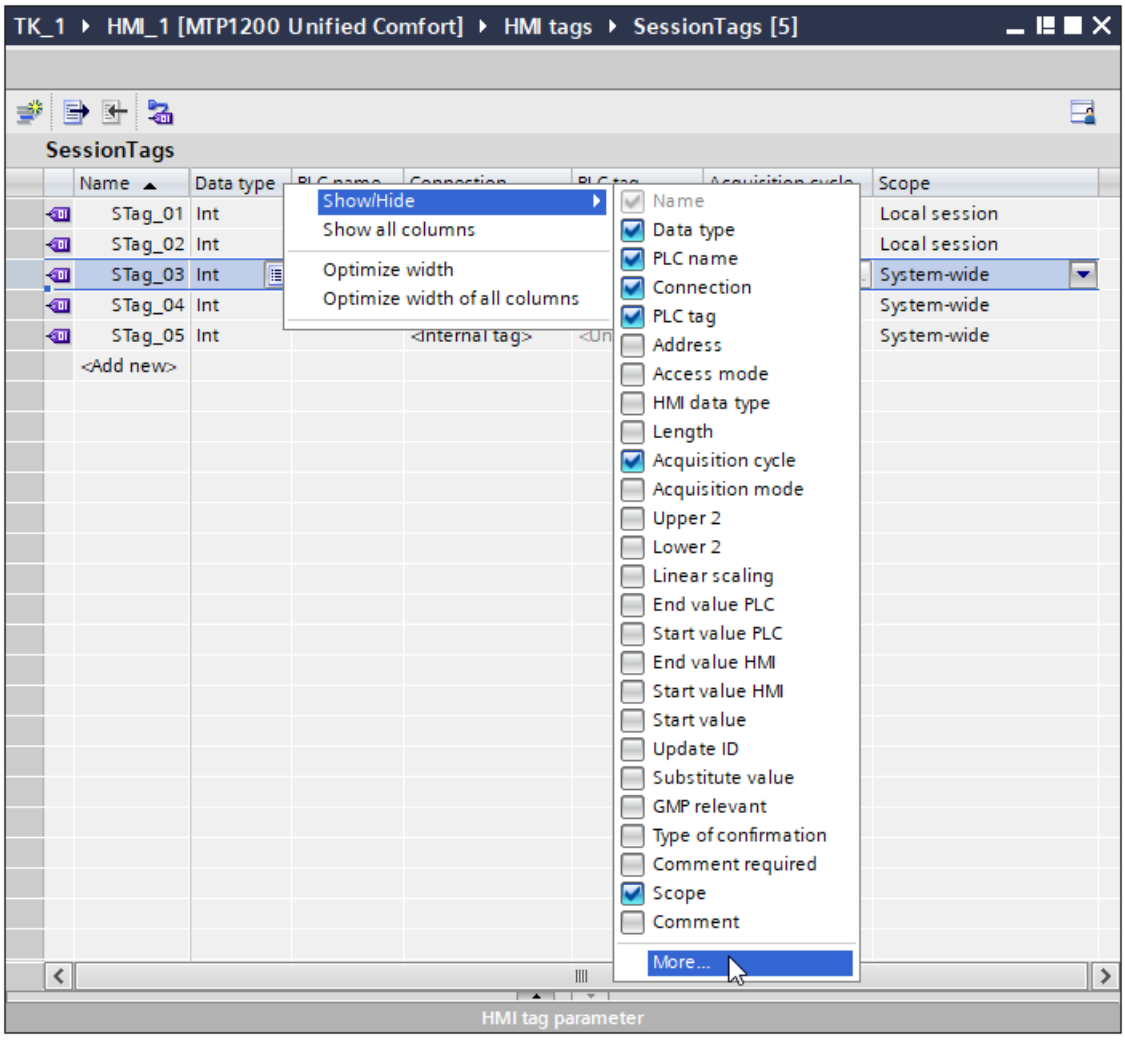
Tags are displayed in tag tables.

Here you create tags, edit their properties and export them.

Adapting the view of the table

The shortcut menu of a column header allows you to

- adjust the width of individual or all columns
- show or hide individual columns



To arrange columns, drag a column header to a different position.

To have your individually designed view displayed again the next time you call up the tag table, click "Save window settings".

To insert a tag above a selected line, click "Lines above ...".

To sort the Table, click on a column header.

See also

Creating external tags (Page 626)

Creating internal tags (Page 629)

Creating OPC tags (Page 628)

5.2.2 Creating external tags

Introduction

You can access an address in the PLC via a PLC tag using an external tag. The following options are available for addressing:

- Symbolic addressing
- Absolute addressing

If possible, use symbolic addressing when configuring a tag. Symbolic addressing enables high-performance data access and is therefore less prone to errors. The system monitors the assignment of the storage address and the locations of use are automatically updated when changes occur.

You create tags either in the standard tag table or in a tag table you defined yourself.

Requirement

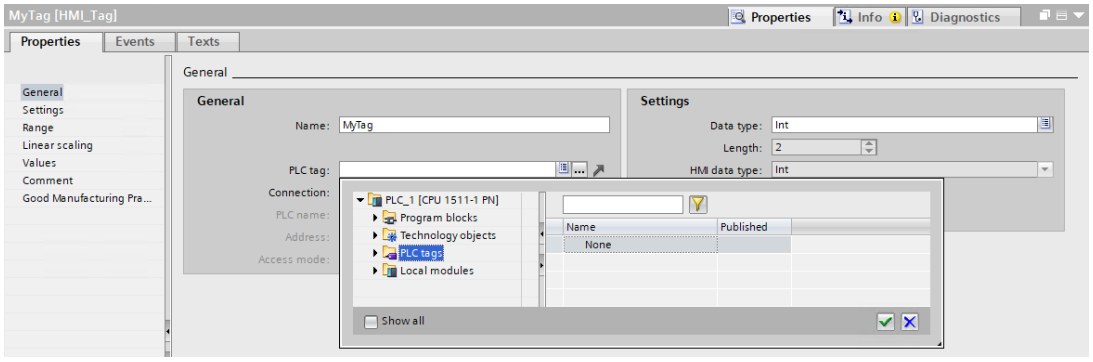
- The project is open.
- A connection to the PLC is configured.
- The Inspector window is open.

Procedure

To create an external tag, proceed as follows:

1. Open the "HMI tags" folder in the project tree and double-click the standard tag table. The tag table opens.
   Alternatively, create and then open a new tag table.
2. In the "Name" column, double-click "Add" in the tag table.
   A new tag is created.
3. Select the "Properties > Properties >General" category in the Inspector window and, if required, enter a unique tag name in the "Name" field.
   The tag name must be unique throughout the device.

4. Select the connection to the required PLC in the "Connection" box.
   If the connection you require is not displayed, you must first create the connection to the PLC. You create the connection to a SIMATIC S7 PLC in the "Devices & Networks" editor. You create the connection to external PLCs in the "Connections" editor.
   If the project contains the PLC and supports integrated connections, you can also have the connection created automatically. To do this, when configuring the HMI tag, simply select an existing PLC tag to which you want to connect the HMI tag. The integrated connection is then automatically created by the system.
5. If you are working with an integrated connection, click the ... button in the "PLC tag" field and select an already created PLC tag in the object list. Click  to confirm the selection.



   Alternatively, use the autocomplete to select a PLC tag for an integrated connection.
   If you select a PLC tag from the autocomplete list, it is entered in the input path. The elements of the PLC tags are displayed in the autocomplete list. If you have selected a PLC tag that can be connected to the HMI tags, the PLC tag is connected to the HMI tags.
6. If you are working with a non-integrated connection, enter the address from the PLC in the "Address" field. To enter the address, make sure that the access mode "Absolute access" is configured.
   The "PLC tag" field remains empty.
7. Configure the other properties of the tag in the inspector window.
   You can also configure all tag properties directly in the tag table. To view hidden columns, activate the column titles using the shortcut menu.

Tips for an efficient procedure

- You also create new tags directly at the location of use, for example, on an I/O field. To do this, click "Add new" in the object list.
  You then configure the new tag in the Inspector window.
- You can also create external HMI tags by dragging and dropping data block elements or global PLC tags into an HMI tag table.

Result

An external tag has been created and linked to a PLC tag or an address in the PLC.

See also

Basics of tags (Page 607)

External tags (Page 610)

Addressing external tags (Page 612)

5.2.3 Creating OPC tags

Requirement

- A connection to an OPC UA server has been set up.
- When the connection to the OPC UA server is secured via a certificate, the application instance certificate of the WinCC Unified clients must be in the "Trusted" folder of the OPC UA server.

Procedure

To create an external tag, proceed as follows:

1. Open the "HMI tags" folder in the project tree and double-click the tag table. The tag table opens.
   Alternatively, create and then open a new tag table.
2. In the "Name" column, double-click "Add" in the tag table.
   A new tag is created.
3. Select the "Properties > Properties >General" category in the Inspector window and, if required, enter a unique tag name in the "Name" field.
   The tag name must be unique throughout the device.
4. If required, select the "Display name" field to enter a name to be displayed in runtime.
5. Select the connection to the desired OPC UA server in the "Connection" field.
   If the connection you require is not displayed, you must first create the connection to the OPC UA server. You establish the connection to an OPC UA server in the "Connections" editor.
6. Click in the "Address" field and follow these steps:
   - In the left area, browse through the address space of the OPC UA server to the node below which the required tag is located.
   - Select the tag in the right area.
   - Confirm your selection.

Result

The new HMI tag is connected to the tag of the OPC UA server. It is available for configuring HMI screens.

After loading the project into a Runtime:

- Runtime has read and write access to the connected OPC UA tag.
- Changes to the tag value on the OPC UA server are automatically passed on to the runtime.

See also

Defining connection settings to the OPC UA server (Page 7045)

5.2.4 Creating internal tags

Introduction

You must at least set the name and data type for internal tags. Select the "Internal tag" item, rather than a connection to a PLC.

For documentation purposes, it is a good idea to enter a comment for every tag.

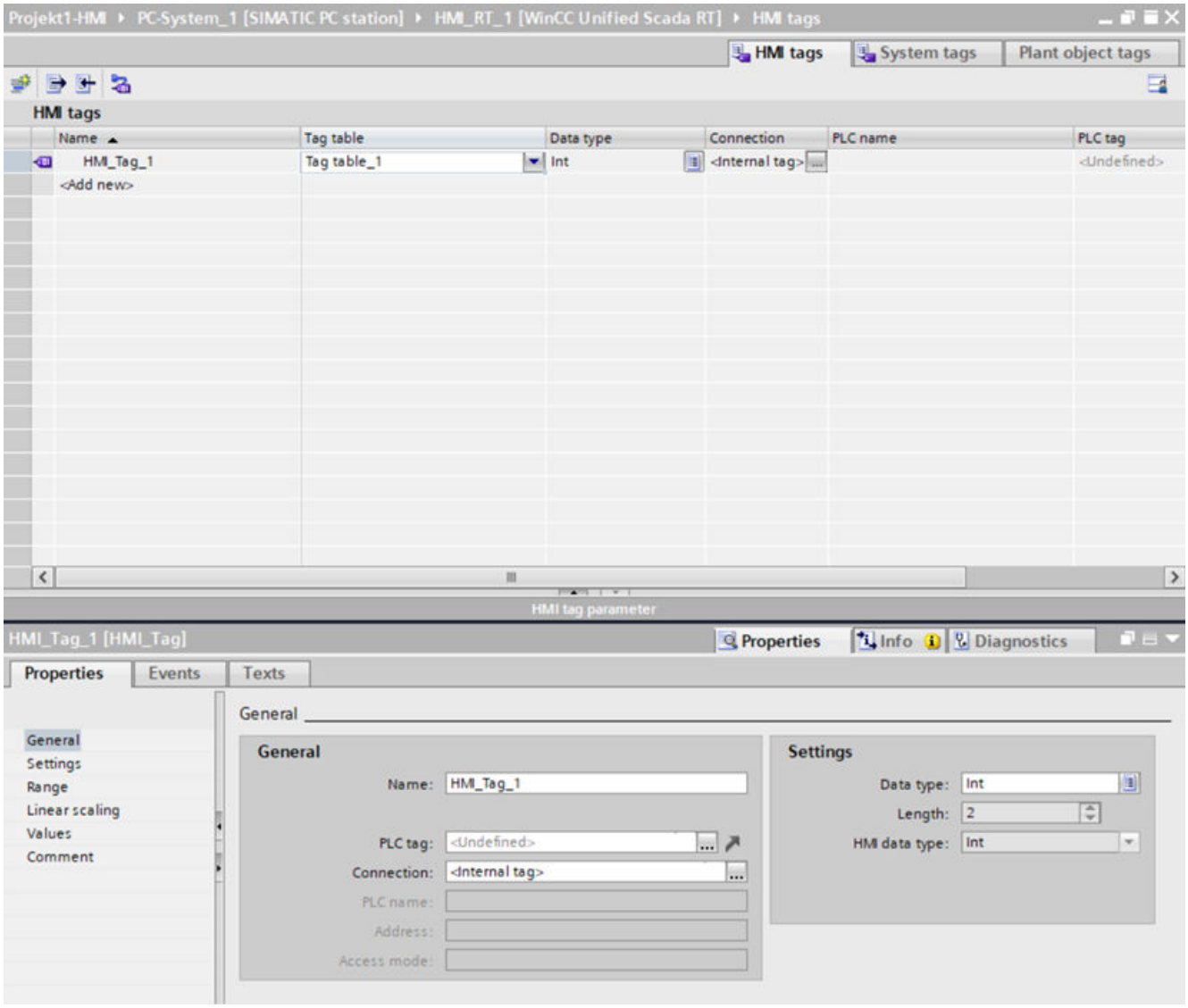
Procedure

1. Open the "HMI tags" folder in the project tree and double-click the entry "Standard tag table". The tag table opens.
   Alternatively, create and then open a new tag table.
2. In the "Name" column, double-click "Add" in the tag table. A new tag is created.
3. If the Inspector window is not open, select the "Inspector window" option in the "View" menu.
4. Select the "Properties > Properties >General" category in the Inspector window and, if required, enter a unique tag name in the "Name" field.

Note

This tag name must be unique throughout the project. The tag name must not contain the special characters line feed, carriage return or quotation marks.

5. If required, select the "Display name" field to enter a name to be displayed in runtime.
   The name to be displayed is language-specific and can be translated for the required runtime languages. The display name is available for Basic Panels, Panels and Runtime Advanced.



6. Select "Internal tag" as the connection in the "Connection" field.
7. Select the required data type in the "Data type" field.
8. In the "Length" field, you can specify the maximum number of characters to be stored in the tag in accordance with the selected data type.
   The length is automatically defined by the data type for numerical tags.
9. As an option, you can enter a comment regarding the use of the tag. To do so, click "Properties > Properties > Comment" in the Inspector window and enter a text.

	Tips for effective configuring
• You also configure the tag properties directly in the tag table. To view hidden columns, activate the column titles using the shortcut menu. • You also create new tags directly at the location of use, for example, on an I/O field. To do this, click "Add new" in the object list. You then configure the new tag in the Inspector window.	

Result

An internal tag is created. You can now use this in your project.

In additional steps you can configure the tag, for example, by setting the start value and limits.

See also

Basics of tags (Page 607)

Internal tags (Page 615)

Configuring discrete alarms (Page 734)

Configuring analog alarms (Page 738)

5.2.5 Configuring multiple tags

Introduction

In a tag table, you create a large number of identical tags by automatically filling the rows of the table below a tag. The tag names are incremented automatically when filling in automatically.

You can also use the auto fill function to fill table cells below a tag with a single tag property and thus modify the corresponding tags.

If you apply the automatic filling in to already filled cells of a tab table, you will be asked whether you want to overwrite the cells or insert new tags.

If you do not want to overwrite already configured tags, activate insert mode. Activate insert mode by keeping the <Ctrl> key pressed during insertion. Already existing entries in the tag table are moved down if insert mode is activated.

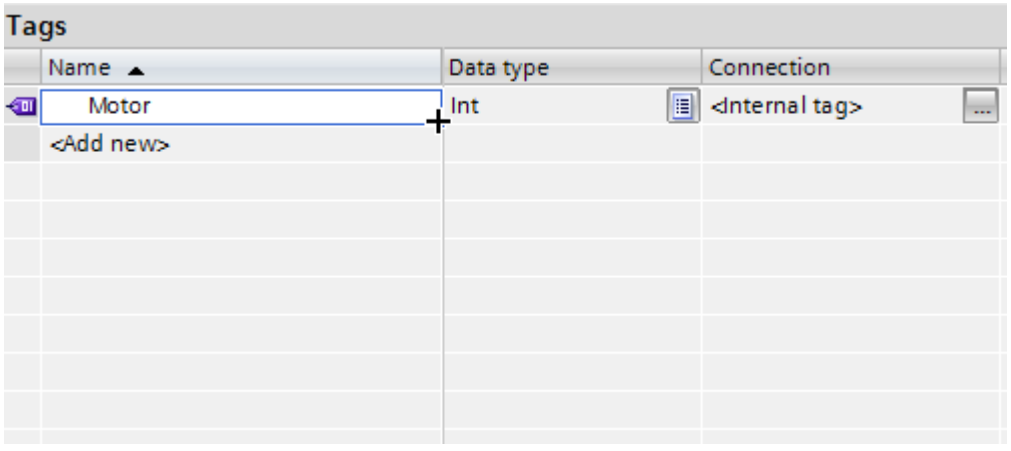
	Tips for effective configuring
You can configure multiple tags simultaneously and use them in the screen. If you use drag and drop to drag multiple tags from the detail view to the screen, an I/O field is created for each tag that is connected to the respective tag.	

Requirement

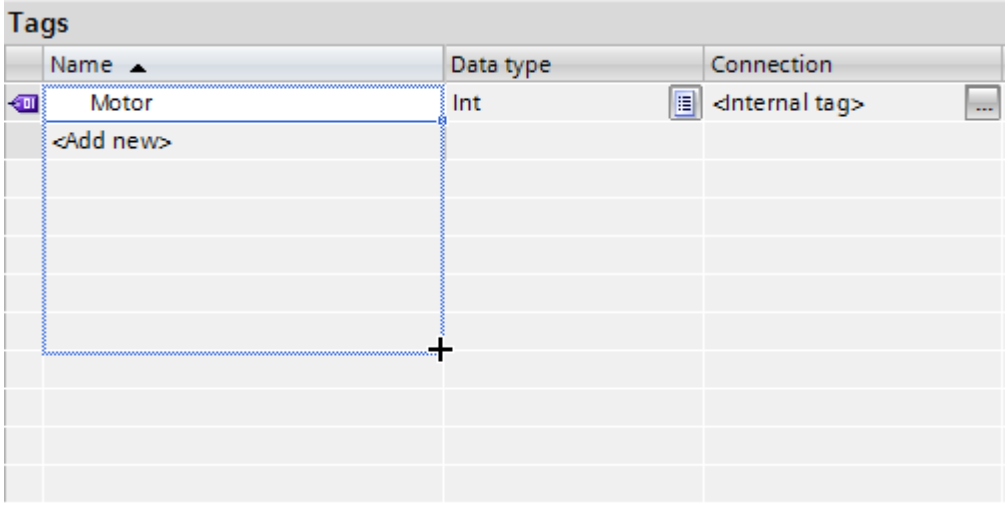
- The project is open.
- A tag table is open.
- The tag which is to serve as a template for other tags is configured.

Procedure

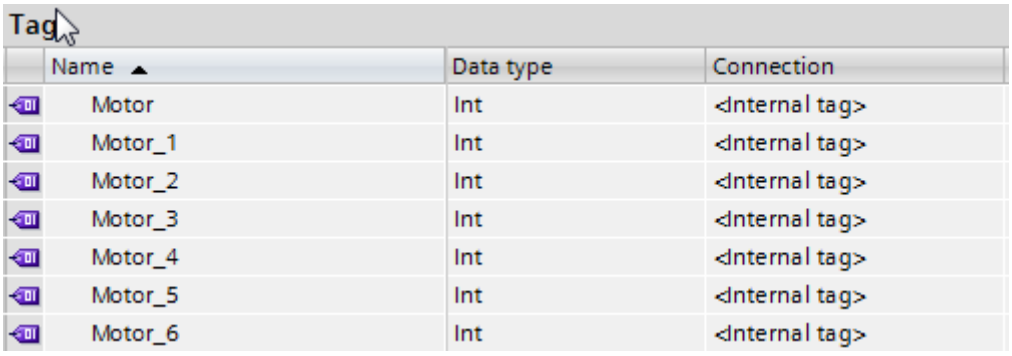
1. If you want to create new tags, mark in the "Name" column the tag that should be used as a template for the new tags.
   If you want to copy a property of a tag to the tags below it, select the cell which contains this property.
   The selected cell will be highlighted in color and a small blue square will appear in its bottom right corner. When you move the mouse over this square, the cursor will change to a black cross.



2. Hold down the mouse button and drag this square over the cells below that you wish to fill automatically.
   The marking will be extended to cover this area.



3. Now release the mouse button. All of the marked cells are filled automatically.
   New tags will be created in all empty rows in the marked area.



4. In the tag table, select all the tags that you want to configure at the same time.
   If the selected property is identical for all the tags, the setting for this property will appear in the Inspector window.
   The associated field will remain blank otherwise.
5. You can define the shared properties in the Inspector window or directly in the tag table.

Result

Depending on which cells were selected, the function may automatically fill individual properties or create and configure new tags.

5.2.6 Adapting the data type of a tag

Introduction

When you create a tag, you assign one of the possible data types to the tag. This data type depends on the type of data for which you would like to use the tag.

The data types available depend on the connected communication partner, such as a PLC.

Note

If you modify the data type of an existing, external tag, the previously defined tag address is identified as invalid. This reason for this is that the PLC address changes when the data type is modified.

Format adjustment

WinCC sets the data type of an external tag according to the data type of the connected PLC tag. If the data type of the PLC tag is not available in WinCC, a compatible data type is automatically used at the HMI tag. As required, you can specify that WinCC uses a different data type and converts the format of the data type of the PLC tag and the data type of the HMI tag.

In WinCC, you have access to the following data types:

HMI data type	Description	Value range
Bool	Binary tag	0 to 1
SInt	Signed 8-bit value	-128 ... +127
USInt	Unsigned 8-bit value	0 ... 255
Int	Signed 16-bit value	-32768 ... +32767
UInt	Unsigned 16-bit value	0 ... 65535
DInt	Signed 32-bit value	-2147483648 ... +2147483647
UDInt	Unsigned 32-bit value	0 ... 4294967295
LInt	Signed 64-bit value	-9223372036854775808 to +9223372036854775807
ULInt	Unsigned 64-bit value	0 to 18446744073709551615
Real	32-bit floating-point number IEEE 754	+-3.402823e+38
LReal	64-bit floating-point number IEEE 754	+-1.79769313486231e+308
Byte	Bit array of 8 bits	8-bit
Word	Bit array of 16 bits	16-bit
DWord	Bit array of 32 bits	32-bit
LWord	Bit array of 64 bits	64-bit
String	Text tag, 8-bit character set	-
WString	Text tag, 16-bit character set	-
WChar	Text tag	-
DateTime	Date/time format	01.01.1601 00:00 ... 31.12.9999 23:59:59
LTime	Signed 64-bit integer value	-9223372036854775808 ... +9223372036854775807 100ns

For format adaptation, select the desired PLC data type at the respective external tag. The suitable standard data type is then selected automatically in the "HMI data type" field for use in WinCC. Change the format adaptation as required.

Data types without format adaptation

The data types are shown 1:1 without format adaptation.

SIMATIC S7-300/400 data types without format adjustment

PLC data type	Description
Bool	No format adaptation
String	No format adaptation

5.2.7 Defining the acquisition cycle for a tag

Introduction

The value of an external tag can be changed in runtime by the PLC to which the tag is linked. To ensure that the HMI device is informed of any changes in tag values by the PLC, the values must be updated on the HMI. The value is updated at regular intervals while the tag is displayed in the process screen or is logged. The interval for regular updates is set with the acquisition cycle.

Requirement

- You have created the tag for which you want to define an acquisition cycle.
- The Inspector window with the tag properties is open.

Procedure

To configure an acquisition cycle for a tag, follow these steps:

1. In the Inspector window, select "Properties > Properties > Settings".
2. Select the acquisition mode for the tag:
   - "Cyclic in operation": The tag is updated at regular intervals as long as it is displayed or archived in the screen.
   - "On demand": Updating is performed only on demand, for example, with the "UpdateTag" system function or a script.
3. If you have selected the "Cyclic in operation" acquisition mode, specify the desired cycle time in the "Acquisition cycle" field.
   You have the option of selecting a user-defined cycle.

Note

For structured HMI tags, the acquisition mode can be selected for the respective structured HMI tag as well as their individual subordinate elements.

When the acquisition mode of structured HMI tags is changed, it is applied to all subordinate elements. This means changing the acquisition mode may overwrite subordinate elements.

Tips for an efficient procedure

Configure the acquisition cycle directly in the work area of the tag table. To view hidden columns, activate the column titles using the shortcut menu.

Result

The configured tag is updated in runtime with the selected acquisition cycle.

See also

Updating the tag value in runtime (Page 617)

Defining cycles (Page 224)

5.2.8 Specify tag persistency

Introduction

In Runtime Unified you have the option to define tag persistency for internal tags. The last values of these persistent tags are used after Runtime has been started.

A separate database, in which the last values of the persistency tags are stored, is used for tag persistency. These values are available again after restarting runtime or restarting or switching off the HMI device.

You activate tag persistency in the runtime settings of the respective HMI device by specifying the database storage location.

Note

Complete loading of a runtime project

Tag persistency can only be used when you enable the option "Keep current values of tags and pending alarms in runtime" in the "Load preview" dialog.

When you disable the option "Keep current values of tags and pending alarms in runtime" in the "Load preview" dialog, the database for the tag persistency is emptied. The persistency tags apply the defined start value. If no initial value is defined, the valid default value is used for the data type.

Requirement

- The tag for which you want to set the persistency is created.
- The Inspector window with the properties for this tag is open.

Procedure

1. In the Inspector window, select "Properties > Properties > Settings".
2. Activate the option "Persistency for internal tags" under "Persistency".
3. Download the full project.

Result

The last value of the configured tag is stored in the tag persistency database.

See also

Creating internal tags (Page 629)

Initial download of a project (Page 7185)

Storage system (Page 7173)

Storage system (Page 7138)

5.2.9 Defining limits for a tag

Note

Limitation

Tags with the following data types do not support limits:

- Array and array elements
- Byte
- Char
- DWord, LWord and Word

Introduction

For numerical tags, you can specify a value range by defining a low and high limit as well as the threshold values.

Requirement

- The tag for which you want to set the limits is created.
- The Inspector window with the properties for this tag is open.

Procedure

To define limits for a tag, proceed as follows:

1. In the Inspector window, select "Properties > Properties > Range". If you want to define one of the limits as a constant value, select "Constant" using the button. Enter a number in the relevant field.
   If you want to define one of the limits as a tag value, select "HMI tag" using the button. Use the object list to define the tag for the limit.
2. To set additional limits for the tag, repeat step 1 with the appropriate settings.

	Tips for an efficient procedure
You also configure the limits directly in the tag table. To view hidden columns, activate the column titles using the shortcut menu.	

Result

You have set a value range defined by the limits for the selected tag.

See also

Limits and start values of a tag (Page 618)

5.2.10 Specify "Local session" scope

Introduction

Internal tags apply "system-wide" by default.

The scope of an internal tag can be changed to "Local session". In a multi-user environment, session-related data is processed independently in each local session.

The use of local session tags in Unified Collaboration and in the web client is supported.

Example scenarios for local session tags

- Individual navigation in screen windows or in different menu structures
- Session-related blocking/unblocking of the user
- Session-related position, alignment and rotation of objects in a screen.

Restrictions

Note

A configuration of tags that contradicts the restrictions, leads either to an error in the engineering system or to an error in runtime.

The "Local session" scope

- is only available for internal tags.
- is only available for tags with elementary data type. Arrays or structure tags cannot be limited to a local session.

Local session tags

- are only accessible within the respective user session
- can only be used in local scripts. Using it in a global script will result in an error.
- cannot be used as a trigger of a task
- cannot be used as a trigger of alarm or for dynamization of an alarm list
- cannot be used to dynamize objects through a resource list
- cannot be used to specify a limit value, substitute value or start value of another tag

- cannot be used as logging tag
- cannot be logged
- do not support the properties
  - GMP (Good Manufacturing Practice)
  - Persistence
- cannot be used in Openness applications

End of a session

- The values of the local session tags are not saved and are lost.

Procedure

1. Open a HMI tag table.
2. Display the "Scope" column.
3. Create an internal tag or select an already created internal tag.
4. In the "Scope" column, select the "Local session" entry.
   You can also make this setting in the Inspector window under "Properties > Properties > Settings".

Result

You have created a tag that can take on different values in different sessions.

This tag can, for example, be addressed by the same script in different sessions. The dynamization of an object can be controlled by the same tag, which takes different values in different sessions.

Changing the version of a device

- Delete the local session tags before changing the device/device version
- Create the tags again after the change.

Note

Local session tags only as of V18

If the version of the device is smaller than V18, the tag automatically becomes "system-wide" because the limited scope is not supported.

5.2.11 Synchronizing tags

Introduction

To synchronize the PLC and HMI tags, WinCC offers the following options:

- Synchronizing tags with or without name matching between PLC and WinCC
  The following options are available for this:
- Link tags with addresses in the PLC
  This procedure is suitable, for example, if changes were made to the connection between the HMI device and the PLC and the tag connections were lost in the process. The function can also be used if you have configured the control program and HMI project separately.

Requirement

- External HMI tags have been created.
- PLC tags have been created.
- An HMI connection to a PLC in the project has been established.

Procedure

To synchronize HMI tags with PLC tags, follow these steps:

1. In the project tree, select the directory that contains the tags in question.
2. Select "Synchronize with the PLC tag" from the shortcut menu.
   A dialog opens.

3. Select the option you want to use.
   If you want to synchronize the tags without name matching, disable "Replace WinCC tag name with PLC tag name".
   If you want to reconnect HMI tags with absolute access, select "Data type and absolute address match".
4. Confirm with "Synchronize".
   The system searches for a suitable PLC tag according to the selected option.

Result

The external HMI tags are synchronized with the PLC tags.

If you have selected the option "Data type and absolute address match", the tag connection is established as soon as a suitable PLC tag is found.

If you have selected a different option, the WinCC tags are updated accordingly and the PLC tag names are applied in WinCC.

See also

Settings for tags (Page 7140)

Settings for tags (Page 7175)

5.2.12 Importing and exporting tags

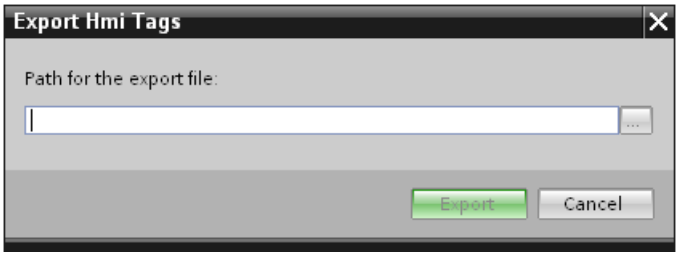
Introduction

WinCC gives you the option of exporting tags to an .xlsx file and reimporting them into the project once you are done editing them. You export and import tags in the "HMI Tags" editor

For importing the tags, the xlsx import file must be structured according to the requirements. You will find more detailed information on the import file under "Export and import of tags (Page 622)".

Exporting tags

1. Click the  button in the "HMI Tags" tab.
   The "Export" dialog box opens.



2. Click "..." and specify in which file the data are saved.
3. Click "Export".
   The export will start.

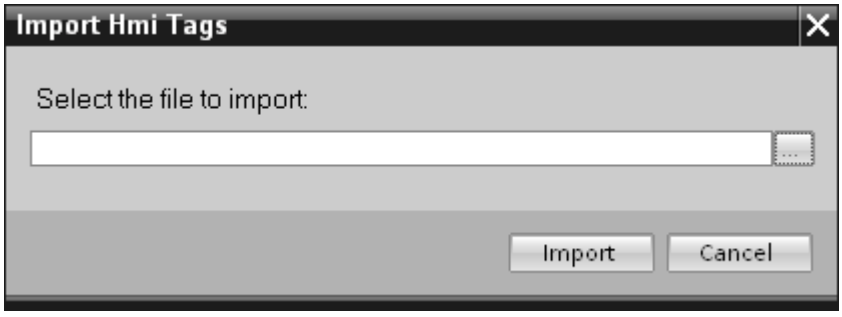
Note

It is not possible to export HMI tags of the data type "UDT" which contain structured elements via Excel for subsequent editing.

After export, only the higher-level HMI tag appears in Excel. Its lower-level elements cannot be edited.

Importing tags

1. Click "HMI tags" in the project tree.
2. Double-click "Show all tags". The "HMI tags" editor opens.
3. Click  in the "HMI Tags" tab.
4. The "Import" dialog box opens.



5. Click "..." and select the file that you want to import.
6. Click "Import".
   The import starts

Result

The relevant tags have been created in WinCC. Alarms relating to the import operation are displayed in the output window. A log file is saved in the source directory of the import files. The log file has the same name as the respective import file but with the "*.xml" extension.

Check when importing the data whether there are any links to objects, for example, dynamic parameters such as tags.

- If an object with the same name already exists, the existing object is used.
- If no object of the same name yet exists, create an object with the relevant name or create a new link.

Note

The syntax of the import file is checked during xlsx file import. The meaning of the properties or dependencies between the properties is not checked.

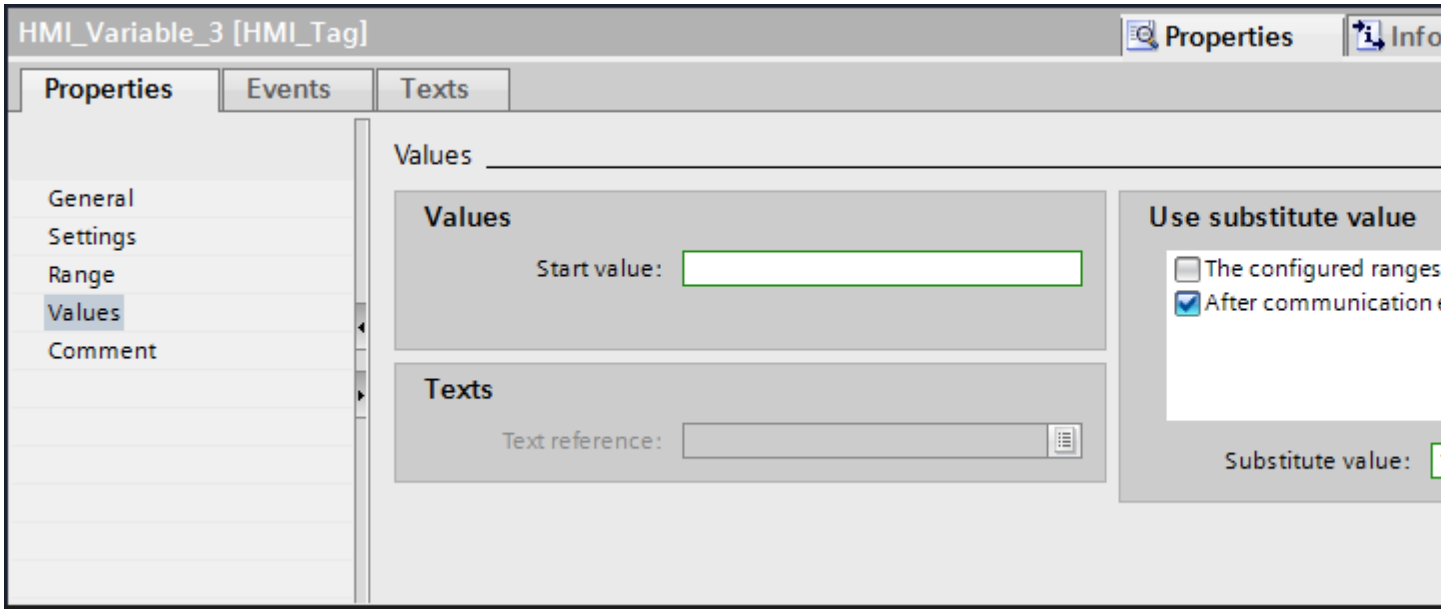
5.2.13 Defining a substitute value

Introduction

You can define a specific value as a substitute value for a tag.

In the "Use substitute value" area you define when WinCC should use this substitute value. The current process value is then not accepted from the automation system.

You can define a substitute value for the following cases:



- The configured ranges have been violated
  If you have set limit values for the tag, WinCC sets the substitute value as soon as the process value violates a limit.
- In the event of a communications error
  WinCC sets the substitute value when the connection to the automation system is disturbed and there is no valid process value.

Requirement

- The tag table is open.
- The Inspector window with the tag properties is open.
- The HMI tag is linked to a PLC tag

Procedure

To configure a substitute value, follow these steps:

1. Select the desired tag in the tag table, and select "Properties > Properties > Values" in the Inspector window.
2. In the "Use substitute value" segment, select when you want WinCC to use this substitute value in the tag.
3. Enter the required substitute value in the "Substitute value" field.

Result

The configured tag is populated with the substitute value in runtime once the selected condition is fulfilled.

Note

If you have set a high or low limit in an I/O field, you cannot enter any value outside these limits.

WinCC ignores incorrect entries and therefore does not set a substitute value. The substitute value is only set by WinCC when an incorrect process value is read.

5.2.14 Connecting a tag to another PLC

Introduction

In WinCC, you can change the PLC connection of a tag at any time. This is needed when you change the configuration of your plant, for example.

Depending on the PLC selected, you may need to modify the configuration of the tag. The tag properties which must be changed will be highlighted in color.

Requirement

- The external tag, whose connection you wish to change, must already exist.
- The connection to the PLC must already exist.
- The Properties window for this tag is open.

Procedure

To change the PLC connection of a tag, proceed as follows:

1. In the Inspector window select "Properties > Properties > General."
2. Select the new connection in the "Connection" field.
   The tag properties that you must change will be highlighted in color in the tag table and in the Inspector window.
3. Change all highlighted properties of the tag to suit the requirements of the new PLC.

Result

The external tag is connected to the new PLC.

5.3 Configuring user data types

5.3.1 Creating an HMI user data type

Introduction

You create an HMI user data type in the project library.

Requirement

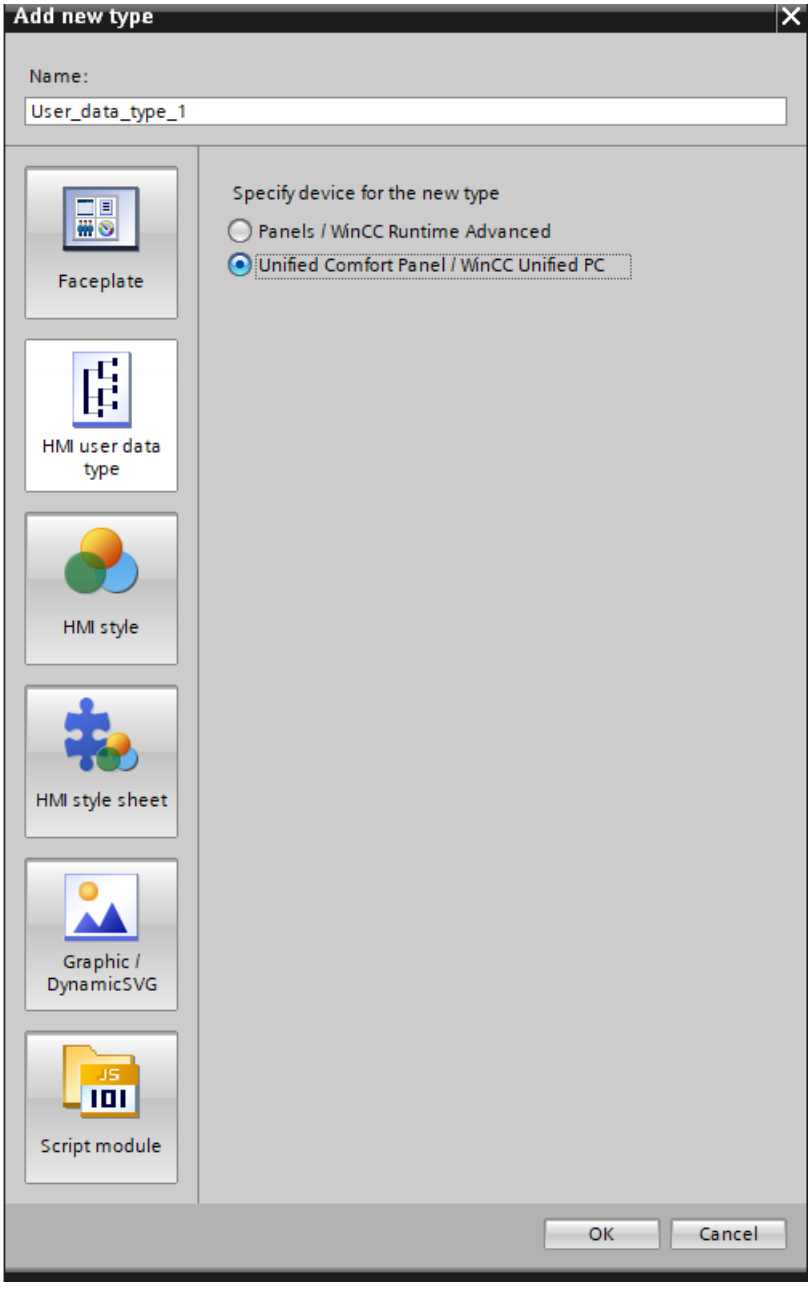
- A project is open.
- The "Libraries" task card is displayed or the library view is open.

Procedure

To create an HMI user data type, follow these steps:

1. Click the "Libraries" task card.
2. Click the "Project library" entry.
   The folder structure of the project library is open.

3. Click "Add new type" in the "Types" folder.
   A dialog opens.



4. Click the "HMI user data type" button in the dialog box.
5. In the "Specify device for the new type" area, select the HMI device on which the HMI user data type is used.
6. Enter a descriptive name in the "Name" field.
7. Click "OK" to apply your settings. The HMI user data type is created. The library view opens.

Result

An HMI user data type with the configured properties is created. Version 0.0.1 of the HMI user data type is created and receives the status "in work".

Create the required user data type elements in the next step.

Before you use the version of an HMI user data type, for example at a tag, the version must be released.

5.3.2 Creating HMI user data type elements

Introduction

You add or delete user data type elements in the "HMI user data types" table.

Requirement

- The library view is open.
- An HMI user data type is created and opened in the editor.

Procedure

1. Select a communication driver for the HMI user data type.
   - If you select the <Internal communication> entry, you can only assign the HMI user data type to the internal tags as the data type.
   - If a connection to a PLC is selected as the communications driver, you can only assign the HMI user data type to tags with a connection to this PLC as the data type.
   - The configured communication driver applies to all user data type elements of an HMI user data type. In an HMI user data type, you can define for each user data type element whether the configured driver is used for the PLC or internal communication.
2. Double-click "Add" in the "Name" column of the table. A new user data type element is added to the HMI user data type.
3. Assign a name.
4. Select the required data type in the "Data type" field.
5. Create as many user data type elements as you need.
6. You configure the user data type elements in the "Properties" group of the Inspector window.

	Tips for an efficient procedure

You can also configure the properties of the user data type elements directly in the table. To view hidden columns, activate the column titles using the shortcut menu.

If you want to transfer a property of a user data type element to other user data type elements, follow these steps:

1. Select the cell that contains the property.
   The selected cell will be highlighted in color and a small blue square will appear in its bottom right corner.
2. When you move the mouse over this square, the cursor will change to a black cross.
3. Hold down the mouse button and drag the cursor up or down over the cells that you want to fill out automatically.
   The marking will be extended to cover this area.
4. Now release the mouse button.
   The "Autocompletion" dialog opens.
5. Select the "Overwrite attributes of the user data type element" option.

Result

You have added elements to version 0.0.1 of the HMI user data type. The version 0.0.1 has the status "in work".

To use the version in the project, release the version.

5.3.3 Adding a PLC user data type to the project library

Introduction

You can use PLC data types in the HMI device.

Requirement

- A project is open.
- A PLC data type is configured in the PLC.
- The project view is open.
- The project library is open.

Procedure

1. Drag the PLC data type to the "Types" directory of the project library with a drag-and-drop operation.
   The "Add type" dialog opens.
2. Specify a type name and a version and click "OK" to confirm.
   In version 0.0.1, the PLC user data type appears in the project library.

Result

You have added the PLC user data type of the project library.

Version 0.0.1 of the PLC user data type was created and receives the status "default".

5.3.4 Managing versions of user data types

Introduction

All HMI user data types have at least one version. When an HMI user data type is created, a version is created at the same time; this version has the status "in work". You can edit the user data type in this status as required. When the editing is complete, you release the version of the HMI user data type. The latest released version is always used at the default.

Requirement

- An HMI user data type is created.
- The HMI user data type has the version 0.0.1. and the status "in work".
- The "Libraries" task card or the library view is open.

Releasing a version

1. Select the version 0.0.1 of the HMI user data type in the project library.
2. Select "Release version" in the shortcut menu.
   A dialog opens.
3. If necessary, change the properties of the version:
   - Enter a name for the type in the "Type name" field.
   - In the "Version" field, define a main and an intermediate version number for the version to be released.
   - To clean up version management of the type, enable "Delete unused type versions from the library".

You have released version 0.0.1 of the HMI user data type.

Editing a version

1. Select, for example, the released version 0.0.1 of an HMI user data type in the project library.
2. Select "Edit type" from the shortcut menu.

The library view opens. The new version 0.0.2 of the HMI user data type was created.

The version has the status "in work".

Restoring the last version of the HMI user data type

The last released version of the HMI user data type is version 0.0.2.

You edit the HMI user data type. A new version of the HMI user data type, 0.0.3, is generated and receives the status "in work".

1. Select the version of the HMI user data type in the project library.
2. Select "Discard changes and delete version" in the shortcut menu.

Alternatively

1. If you have opened a version for editing, click "Discard changes and delete version" in the toolbar.
   The version is deleted.

All changes to the HMI user data type since the last release operation are discarded. The HMI user data type is released again and has version 0.0.2.

Deleting user data type

If you delete a user data type, all instances and master copies of this user data type are deleted as well. The same is true for HMI tags that use an HMI user data type and for PLC user data types.

To delete a user data type, follow these steps:

1. In the project library, under "Types", select the user data type you want to delete.
2. Select "Delete" from the shortcut menu.

See also

Setting a user data type version as default (Page 650)

5.3.5 Setting a user data type version as default

When you add a user data type to a library, use data types from a library, and release or update versions, the highest released version is used as the default. This version has the "default" identifier. Alternatively, you can specify a different released version as default.

Requirements

- You have opened the project library or a global library.
- The desired type version is released.

Procedure

1. Select a version.
2. Right-click to call up the shortcut menu.
3. Select the shortcut menu command "Set as 'default'".

Result

When instantiating, creating, releasing and updating, the default user data type version is used instead of the highest released version. The default user data type version is also used when you create newer user data type versions.

The icon in the "Status" column shows whether the references of the type are consistent with other types:

- Green: The references of the user data type are consistent. The version of the user data type labeled "Default" references the "Default" user data type version of the dependent type.
- Yellow: The references of the user data type are inconsistent. The version of the user data type labeled "Default" does not reference the "Default" user data type version of the dependent type.

5.3.6 Creating tags with a HMI user data type

Introduction

When a tag is created, you assign the version of an HMI user data type as a data type. In the "Tag" editor you can create internal tags or tags with a link to a PLC. A tag provides all HMI user data type versions that use the same communication driver as the tag itself.
You can only use an HMI user data type in combination with a PLC when you have selected absolute addressing.

Requirement

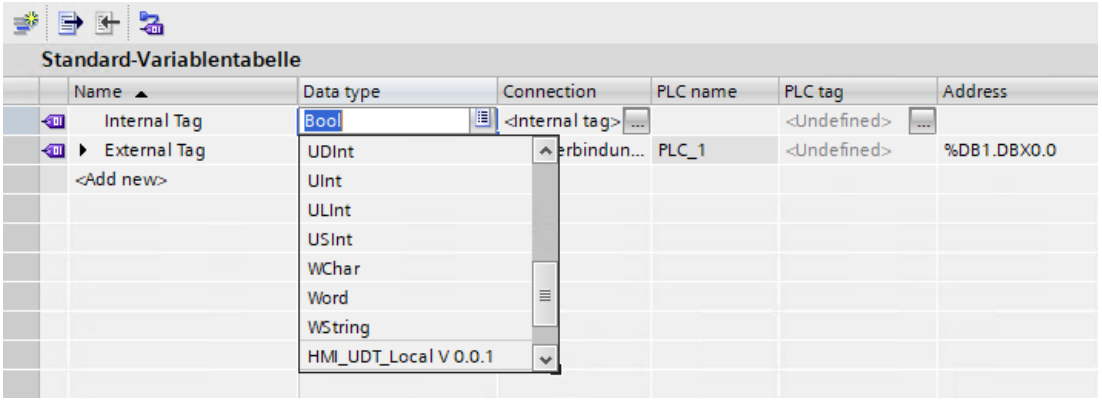
- An HMI user data type with a user data type element is created.
- The HMI user data type is released.
- The tag table is open.
- The Inspector window with the tag properties is open.

Procedure

To create a tag of the "User data type" data type, follow these steps:

1. In the "Name" column, double-click "Add" in the tag table. A new tag is created.
2. Enter a unique tag name in the "Name" field.
3. Select the connection to the required PLC in the "Connection" box.

4. Select the desired version of the HMI user data type in the "Data type" field.
   The selected connection determines which HMI user data types are displayed.
   For internal tags, only HMI user data type versions of the <Internal user data type> type are available.



Note

For tags with a connection to a PLC, only those HMI user data types that have a link to a PLC are available.

5. In the "Address" field, enter the address of the PLC that you want to access with the external tag. The specified address must always point to the start data bit, for example, <DB1.DBX0.0>.

Result

You have created a tag of the "User data type" data type. In additional steps you can configure the tag, for example, by setting the start value and limits.

If you wish to change the properties of an HMI user data type tag, you must change the properties of the HMI user data type elements.

Properties, such as "Start value" and "Linear scaling", can also be changed in the HMI user data type instances used.

5.4 Logging tags

5.4.1 Basics

5.4.1.1 Basics of data logging

Introduction

Data logging is used to collect and log process data from an industrial system. This allows you to analyze error statuses and document your process.

From the archived process data, you extract important business and technical information about the operating status of a plant in an evaluation.

Logging concept

WinCC Unified is optimized for event-driven logging. The process values are saved by configured events and selected processing mechanisms in the data log. Event-driven logging does not depend on the logging mode.

In the past, we have used query-driven logging. This method saved the values at fixed cycles without additional processing in the log. You can map query-driven logging by using the "Cyclic" logging mode and by not configuring any smoothing or compression. The procedure is not recommended.

Event-driven logging offers the following advantages compared to query-driven logging:

- High accuracy
  Note that logging can only be as precise as the data acquisition at the PLC.
- High performance because fewer values must be logged

If you need a query-driven representation of the values for data evaluation, we recommend that you use event-driven logging and then interpolate the missing values.

Configuration

You configure the logging tags in the "Logging tags" tab in the "HMI tags" editor.

You use logging tags in the following controls in Runtime:

- Trend control
- f(x) trend control
- Process control

Use

You can use data logging for the following tasks:

- Early detection of danger and fault conditions
- Increase of productivity
- Increase of product quality
- Optimization of maintenance cycles
- Documentation of process value trends

See also

Log basics (Page 837)

How it works (Page 839)

Storage locations of logs (Page 843)

Creating a data log and an alarm log (Page 848)

Editing log contents with scripts and system functions (Page 850)

Size of a log entry in the data log (Page 654)

5.4.1.2 Size of a log entry in the data log

The size of a log entry depends on the database type used.

The following values result for the data log:

	SQLite	**Microsoft SQL**
Additional memory requirement per segment	Approx. 0.5 MB	Approx. 5 MB
Size of a log entry	Approx. 50 bytes	Approx. 50 bytes

See also

Basics of data logging (Page 653)

How it works (Page 839)

5.4.1.3 Logging modes and logging process

Logging begins when Runtime starts. The process values to be logged are acquired, processed, and saved in the data log. Current or previously logged process values can be output in runtime.

Three logging modes are available for logging tags:

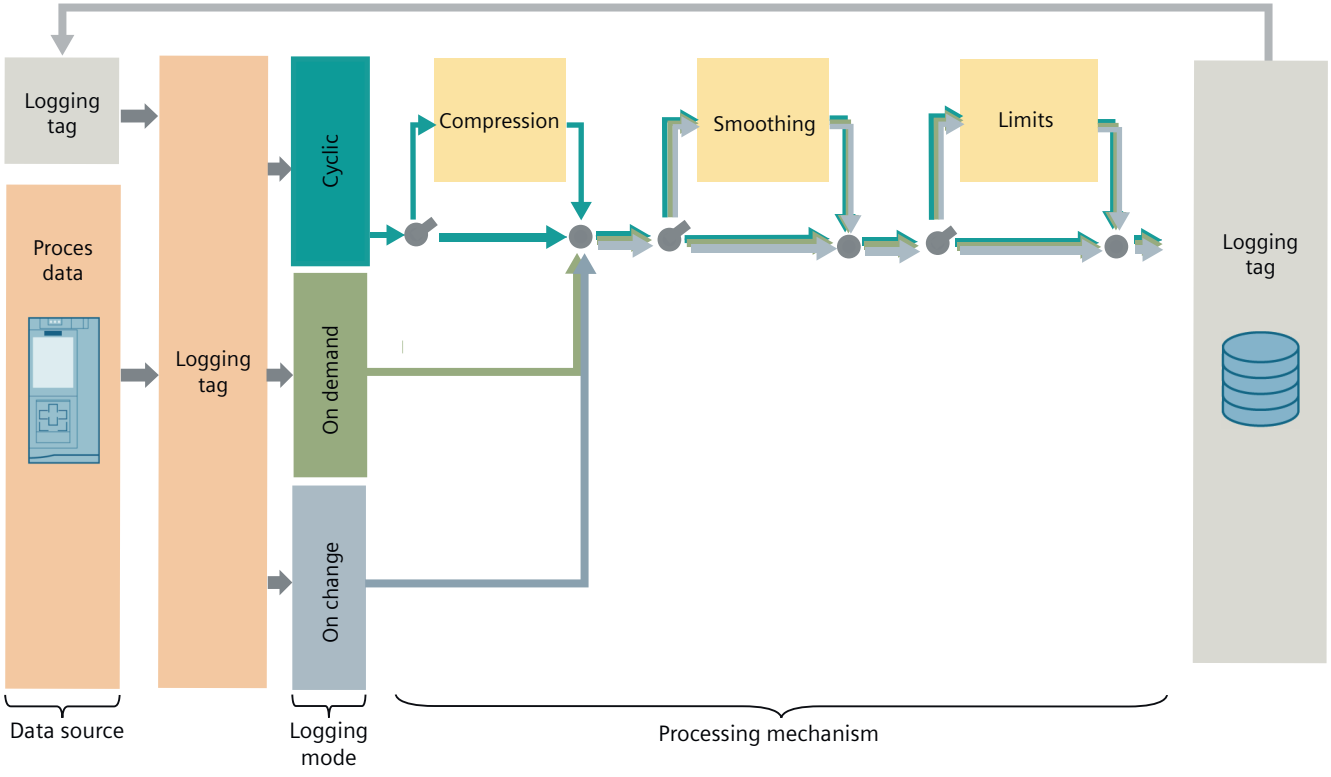
- On change: When the process value changes, it is logged.
- On demand: When the tag trigger is triggered, the process value is logged.
- Cyclic: Logging is carried out according to a defined cycle.

The amount of logged data can be reduced due the following processing mechanisms:

- Limits
- Smoothing
- Compression
  When you select an existing logging tag as source, the values are compressed with the selected compression mode.

Data logging ends when runtime is terminated.

Logging process



"On change" logging mode

When the process value changes, the process value is saved in the data log in the "On change" logging mode while runtime is running. You can also define a tag trigger.

- The acquisition cycle of the external tags controls when the process value is read from the memory of the connected PLC.
  The external tags in WinCC correspond to a process value in the memory of a connected PLC.
- When the process value changes, the runtime component of the logging system processes the process value.
- If you have configured smoothing, the value is calculated accordingly.
- The processed process value is written to the data log.

For defined tag triggers, the tag is also logged when the trigger occurs.

"On demand" logging mode

- The process value is read from the PLC when the request by the tag trigger occurs.
  The external tags in WinCC correspond to a process value in the memory of a connected PLC.
- If you have configured smoothing, the value is calculated accordingly.
- The processed process value is written to the data log.
- With each new request by the tag trigger, the process value is read, processed and logged again from the memory of the connected PLC.

"Cyclic" logging mode

The process values are recorded in constant cycles and stored in the data log. You have the option of using a predefined cycle or a user-defined cycle.

The process value is only logged if it differs from the previous value. If every value is to be logged, no smoothing and no compression must be configured for the logging tag.

Note

Depending on the configuration, the database can grow very quickly. This can occur, for example, when you select a short cycle without smoothing and without compression.

- The acquisition cycle of the external tag controls when the process value is read from the memory of the connected PLC.
  The external tags in WinCC correspond to a process value in the memory of a connected PLC.
- If you have configured smoothing or compression, the value is calculated accordingly.
- The logging cycle determines when the processed process value is written to the log database.

Note

The logging cycle is independent of the acquisition cycle of the external tag. It does not make sense to set the logging cycle shorter than the acquisition cycle.

5.4.2 Configuring logging tags

Introduction

You manage the logging tags at various locations:

- You have the following options in the "HMI tags" editor:
  - Create logging tags
  - Edit logging tags
  - Delete logging tags
- You have the following options in the "Logs" editor:
  - Edit logging tags
  - Delete logging tags

Note

Logging tags are deleted

Logging tags are deleted with the following actions and must be recreated:

- The logging tag is assigned to an array element. The data type of the array is changed.
- The logging tag is assigned to a user data type element. The data type of the user data type element is changed, and a new version is released without updating the associated instances. The new version of the user data type is assigned to the tag manually in the "HMI tags" editor.

Note

Local session tags

Local session tags cannot

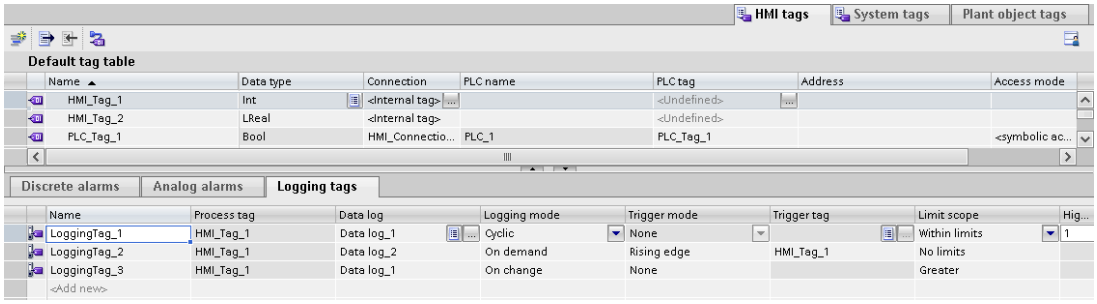
- be used as a trigger
- be used as an logging tag
- be logged.

Requirement

- The "HMI tags" editor is open.
- The logging tag tab is expanded.
- You have created a tag.

Procedure

1. Select an existing tag in the tag table.
2. In the table of the "Logging tag" editor table, double-click "<Add>" in the "Name" column.
   A new logging tag is created.
   The logging tag is linked to the selected tag. The data type of the logging tag corresponds to the data type of the higher-level tag.



3. Assign a data log to the new logging tag. Every logging tag can be assigned to another log.
4. Specify the logging mode.
5. When the "Cyclic" logging mode is set, define the logging cycle and the factor under "Properties > Properties > Cycle".
   The product of logging cycle and factor determines the time interval between two logs. For example, if you define "1 second" as the time basis and "5" as the time factor, the process values are logged every 5 seconds.
   The factor is a multiplier for the duration of the configured logging cycle.
   The factor is used to reduce the number of cycle times. Cycle times that you configure are used for the acquisition cycle and the logging cycles to acquire and then log the measured values. As the different measured values are acquired and logged with different cycle times, different times must also be defined. The factor reduces the number of defined times. This means that times with factor are used several times. A logging cycle of 120 s with factor = 1 corresponds to a logging cycle of 60 s with factor = 2. Both measured values are logged every 120 s.
6. Define the tag trigger depending on the logging mode.
7. Define the limit values.
8. Define the smoothing.
9. If you have selected the "Cyclic" logging mode, define the compression.

	Tips for an efficient procedure
You can also configure the properties of the logging tags directly in the logging tags table. To view hidden columns, activate the column titles using the shortcut menu.	

Copying logging tags

You can copy logging tags within an HMI device. To do this, use the shortcut menu or the shortcuts <Ctrl + C> and <Ctrl + V>.

You copy logging tags, for example, from:

- "HMI tags" editor
- "Logs" editor
- Data source of a control in which logging tags are used

You insert logging tags, for example, at the following locations:

- "HMI tags" editor
- Data source of a control in which logging tags are used

Note

If you hare using cascading logging tags by having defined a source under "Compression", the reference becomes invalid due to the copy process. Redefine the source for the inserted logging tag.

See also

Creating a data log and an alarm log (Page 848)

Configuring tag triggers (Page 660)

Configuring limit values (Page 661)

Configuring smoothing (Page 663)

Configuring compression (Page 667)

5.4.3 Configuring multiple logging tags

Introduction

You can create multiple logging tags at the same time in the "HMI tags" editor. Then you adapt the properties of multiple logging tags by automatic filling. This will shorten the configuration time.

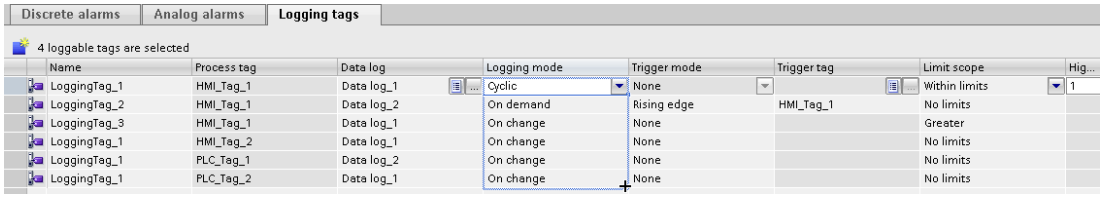
You edit and delete the logging tags in the "HMI tags" editor or in the "Logs" editor.

Requirement

- Multiple tags are configured.
- A tag table is open.

Procedure

1. Select multiple tags of the tag table.
   If you have already configured logging tags, all logging tags that are assigned to the selected tags are displayed.
2. Select the button "Add new logging tag to each selected loggable tag".
   A tag is added to each selected tag that can be archived.
3. When you want to transfer a property of a logging tag to other logging tags, select the cell that contains this property.
   The selected cell will be highlighted in color and a small blue square will appear in its bottom right corner.
4. When you move the mouse over this square, the cursor will change to a black cross.
5. Hold down the mouse button and drag the cursor up or down over the cells that you want to fill out automatically.
   The marking will be extended to cover this area.



6. Now release the mouse button.
   All of the marked cells are filled automatically.

Tips for an efficient procedure

For a better overview, you can sort the columns of the logging tag table in ascending or descending order. To do this, click on the column title. To view hidden columns, activate the column titles using the shortcut menu.

Result

- You have created multiple logging tags.
- You have defined the properties of the logging tags through automatic filling.
  If properties for individual logging tags cannot be applied, the previous values are retained.

5.4.4 Configuring tag triggers

Introduction

When you use the "On demand" and "On change" logging modes, you have the option of defining a tag trigger. In the "On demand" logging mode, you log the tags regardless of the value change, and you can trigger logging, for example, at the end of a process or a shift. When the defined bit of the trigger tag changes according to the defined mode, the tag is logged.

Requirement

- You have created a logging tag.
- The logging mode "On demand" or "On change" is defined for the logging tag.

Procedure

1. Select an existing logging tag in the "Logging tags" tab.
2. In the Inspector window under "Properties > Properties", select the "Tag trigger" area.

   Note

   Local session tags cannot be used as triggers.

3. Specify a trigger mode:
   - None: The tag trigger is not used.
   - Rising edge: When the bit changes from 0 to 1, the trigger is triggered.
   - Falling edge: When the bit changes from 1 to 0, the trigger is triggered.
   - Rising and falling edge: When the bit changes from 0 to 1 or from 1 to 0, the trigger is triggered.
4. Define the trigger tag.
5. Specify the bit that is to be considered.

Tips for an efficient procedure

You can also configure the tag trigger directly in the logging tags table. To view hidden columns, activate the column titles using the shortcut menu.

Result

You have defined a tag trigger that triggers logging.

See also

Configuring logging tags (Page 657)

5.4.5 Configuring limit values

Introduction

You have the option of archiving tag values outside or within a defined limit scope. To do this, define a valid range and the corresponding limit values. In runtime, the process values are evaluated after the configured limit scope and only the process values within the defined range are logged.

Limit scope

The following limit scopes are available:

Limit scope	Description	Example
No limits	No limit values are defined for logging.	Limit values are not taken into consideration.
Greater	The process values that are greater than the lower limit value are logged.	Low limit =3; Logged values = 4, 5, 6
Less	Only the process values that are less than the high limit are logged.	High limit =6; Logged values = 3, 4, 5
Greater or equal	Only process values that are equal to or greater than the lower limit are logged.	Low limit =3; Logged values = 3, 4, 5, 6
Less or equal	Only the process values that are equal to or less than the high limit are logged.	High limit =6; Logged values = 3, 4, 5, 6
Within limits	Only the process values that are within the two configured limits are logged.	Low limit = 3, high limit = 6; Logged values = 4, 5
Within or equal	Only the process values that are within the two limits or equal to one of the limit values are logged.	Low limit = 3, high limit = 6; Logged values = 3, 4, 5, 6
Outside limits	Only the process values that are outside of the two configured limits are logged.	Low limit = 3, high limit = 6; Logged values = 1, 2, 7, 8
Outside or equal	Only the process values that are outside the two limits or that correspond to one of the limit values are logged.	Low limit = 3, high limit = 6; Logged values = 1, 2, 3, 6, 7, 8

Requirement

- You have created a logging tag.

Procedure

1. Select an existing logging tag in the "Logging tags" tab.
2. In the Inspector window under "Properties > Properties", select the "Limit values" area.
3. Specify the limit values. You have the following options:
   - Use constants.
     Select "Constant" using the selection button.
     Enter a number in the relevant field.
   - Use HMI tags.
     Select "HMI tag" using the selection button.
     Open the selection dialog. Only tags that correspond to the data type of the logging tag are displayed. Local session tags are not permitted.
     Select a tag.
   - Apply the limit values of the associated HMI tag.
     Activate "Use tag limits".
     The limit values of the HMI tag are used and grayed out in the corresponding fields.

	**Tips for an efficient procedure**
You can also configure the limits directly in the logging tags table. To view hidden columns, activate the column titles using the shortcut menu.	

Result

You have specified a value range for the selected logging tag that is defined by the limit values.

See also

Configuring logging tags (Page 657)

5.4.6 Configuring smoothing

Introduction

You can compress the data volume of the logged data using smoothing to reduce the memory space required. Smoothing reduces the noise in the collected data.

The process values are only logged in accordance with certain predefined criteria.

Note

Smoothing

If the value "No smoothing" is set in the properties of the logging tag under "Smoothing > Mode", the values are nevertheless smoothed.

Example:

A logging tag changes its value as follows: "100" > "101" > "101".

Even if "No smoothing" is set in the properties of the tag, the values [100, 101] are logged.

Requirement

- You have created a logging tag.

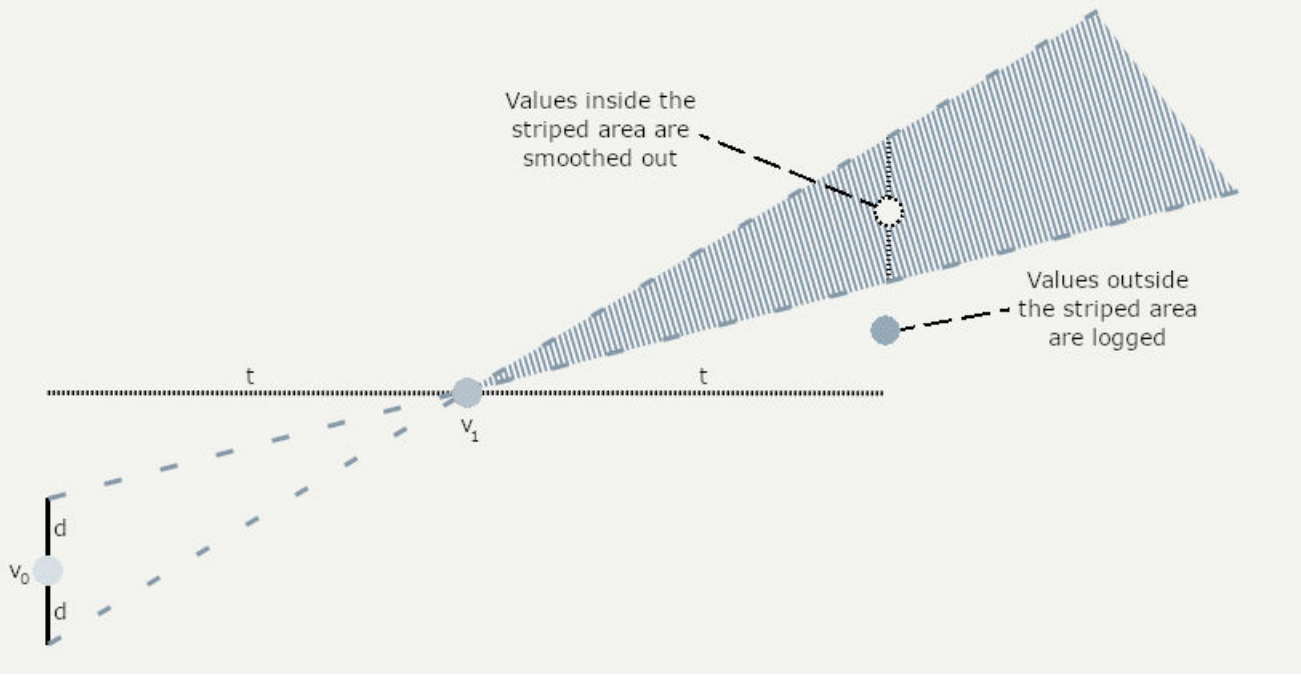
Procedure

Select one of the following modes:

- No smoothing
  The values are logged without smoothing.
- Comparing values
  Values are logged when the value changes and/or the quality code is changed. No logging is triggered when only the time stamp changes.

- Value
  You specify a limit value that defines the maximum permitted distance between the values. All value changes occurring within the defined interval from the last logged value are not logged.

- Relative value
  You specify a percentage deviation that defines the maximum interval between the values. All value changes occurring within the defined interval relative to the last logged value are not logged.
  Example:
  - You define a deviation of 10%, the last logged value is 100.
  - The value 105 is not logged because the value change is less than 10%. On the other hand, the value 130 is logged because the value change is more than 10% and is therefore relevant for logging.
- Swinging door
  The Swinging Door algorithm is a combination of value-based and time-based smoothing. The swinging door algorithm evaluates values on the basis of the defined rate of change and only logs them if the following value is outside the calculated range. The compression rate depends on the maximum tolerated deviation. The deviation is set as an absolute value.



t	Time interval between received values
d	Configured deviation
v_0	Previously logged value
v_1	Current value

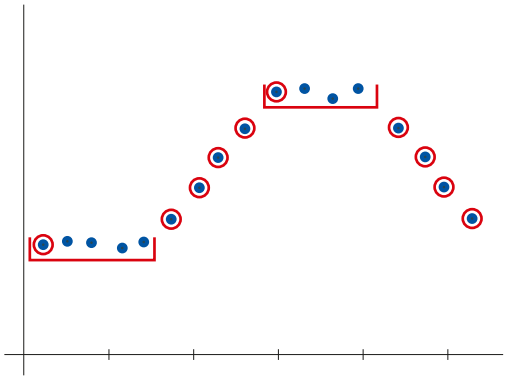
The starting point for calculating the next logging time is the last value logged.
Using the set deviation, you can influence the precision with which the values are saved. The greater the deviation, the fewer values are logged. Values for which "Deviation is within the tolerated deviation" is true are not taken into account.
With the maximum time, you specify the time after which a new value will definitely be logged. This specifies additional reference values for the logged data even if no significant changes occur during this time. With the minimum time, you specify the time interval after which the next value for logging is calculated. All measured values within the minimum time are not logged.

	Tips for an efficient procedure
You can also configure smoothing directly in the logging tags table. To view hidden columns, activate the column titles using the shortcut menu.	

Example – Smoothing with the value

You specify a constant value for the deviation. All values that are within the defined deviation and have not changed significantly are not logged.

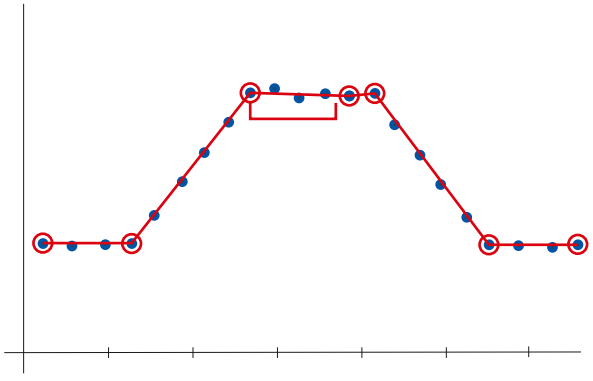
Only the values outside the deviation are written to the log.



Example – Swinging door with deviation and maximum time

You define the deviation and the maximum time after which the next value is written to the log.

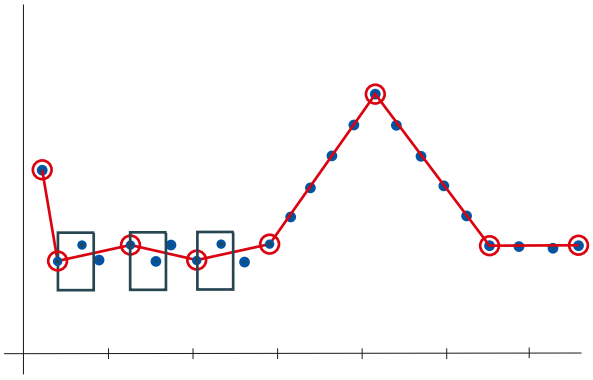
Once the first value has been saved, the following values are evaluated in the pre-defined rate of change. If the value is within the rate of change, it is not logged. If the value is outside the deviation, it is logged. In addition, the values that do not have a significant change compared to the previous logging value are logged at regular defined intervals.



Example – Swinging door with deviation and minimum time

You define the deviation and the minimum time after which the next value is to be evaluated in runtime.

Once the first value has been saved, the next value for logging in runtime is calculated after the preset time. If the value is within the rate of change, it is not logged. If the value is outside the deviation, it is logged.



See also

Configuring logging tags (Page 657)

5.4.7 Configuring compression

Introduction

In the "Cyclic" logging mode, you can compress the data volume of the logged data using compression to reduce the memory space required.

If you have selected "Cyclic" as logging mode, runtime logs the tag values according to the logging cycle, the defined smoothing and the defined compression.

Various compression modes are available.

Supported data types

The following table shows which compression mode supports which data types. You can see from the table in which data type the source data is converted by compaction. A "==" means that the source data type remains unchanged.

	Compression mode								
Data type	**No compression**	**Minimum**	**Maximum**	**Minimum with time stamp**	**Maximum with time stamp**	**Total**	**Average**	**Time average sloped step by step**	**End**
Bool	==	Not supported	Not supported	Not supported	Not supported	Not supported	LReal	LReal	==
SInt	==	==	==	==	==	LInt	LReal	LReal	==
USInt	==	==	==	==	==	ULInt	LReal	LReal	==
Int	==	==	==	==	==	LInt	LReal	LReal	==
UInt	==	==	==	==	==	ULint	LReal	LReal	==
DInt	==	==	==	==	==	LInt	LReal	LReal	==
UDInt	==	==	==	==	==	ULInt	LReal	LReal	==
LInt	==	==	==	==	==	Not supported	Not supported	Not supported	==
ULInt	==	==	==	==	==	Not supported	Not supported	Not supported	==
Real	==	==	==	==	==	LReal	LReal	LReal	==
LReal	==	==	==	==	==	LReal	LReal	LReal	==
String	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
Time	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
DateTime	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
Char	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
Byte	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
Word	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
DWord	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==
LWord	==	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	==

Note

Note the following when configuring the cycle and the compression:

- You can overload the processor if you use short cycles and compression. Make sure that the computing power of the processor is sufficient to calculate the values to be logged for each cycle.
- Depending on the configuration, the database can grow very quickly. This can occur, for example, when you select a short cycle in combination with the smoothing mode "No smoothing" and the compression mode "No compression".

Requirement

- You have created a logging tag.
- The "Cyclic" logging mode is defined for the logging tag.

Procedure

1. Select one of the following modes:

Compression mode	Description
No compression	Every value is logged. Logging receives a time stamp of the interval end. Examples: • If you have not defined any smoothing, every value is logged. • If you have defined the "Compare values" smoothing mode, the value is only logged when the value changes and/or the quality code is changed.
Minimum	The minimum of the values determined within the logging interval, including the start value, is logged. The logging receives a time stamp of the interval start.
Maximum	The maximum of the values determined within the logging interval, including the start value, is logged. The logging receives a time stamp of the interval start.
Minimum with time stamp	The minimum of the values determined within the logging interval, including the start value, is logged. Unlike in the "Minimum" mode, in this mode the logged minimum value receives the time stamp of its occurrence.
Maximum with time stamp	The maximum of the values determined within the logging interval, including the start value, is logged. Unlike in the "Maximum" mode, in this mode the logged maximum value receives the time stamp of its occurrence.
Total	The total of all values determined within the specified logging interval is logged without the start and end values.
Average	The average value of all values determined within the specified logging interval is logged without the start and end values.
Time average sloped step by step	The time-weighted average value of all values determined within the specified logging interval without start and end value is logged.
End	The last value determined within the specified logging interval is logged. The logging receives a time stamp of the interval start. Examples: • If you have not defined any smoothing, the zero value and the quality code "NoData" will be logged when there are no value changes. • If you have defined the "Compare values" smoothing mode, the value is only logged when the value changes and/or the quality code is changed.

Note

Each value is logged when you select "No smoothing" for the smoothing mode in combination with the compression mode "No compression". This will quickly increase the log size.

2. You have the option of defining a delay. The delay value defines the latest possible time up to which the compression value is logged after the end of a logging cycle. If the time stamp of a value is after the delay value, the value is not logged.
3. Under "Source", you can select an existing logging tag whose values are to be compressed with the selected compression mode.

Tips for an efficient procedure

You can also configure compression directly in the logging tags table. To view hidden columns, activate the column titles using the shortcut menu.

Result

You have configured compression for a logging tag.

See also

Configuring logging tags (Page 657)

5.5 Displaying tags

5.5.1 Basics

5.5.1.1 Outputting the tag values

Overview

With WinCC you can output tag values in the HMI screen with different screen objects and change them.

- The I/O field is used for the input and output of process values.
- Bars are used for graphic display of the process values in form of a scale.
- Sliders are used for the input and output of process values within a defined range.
- The gauge is used to display the process values in form of an analog gauge.

In runtime you can also output tag values as table or as trend. You can use either process values or logged values as source for the tag values.

- Use a trend for the graphic display of tag values. Trends allows you to display the change in motor temperature, for example.
- Use a table to compare tag values. In the table you can, for example, compare fill levels of supply tanks.

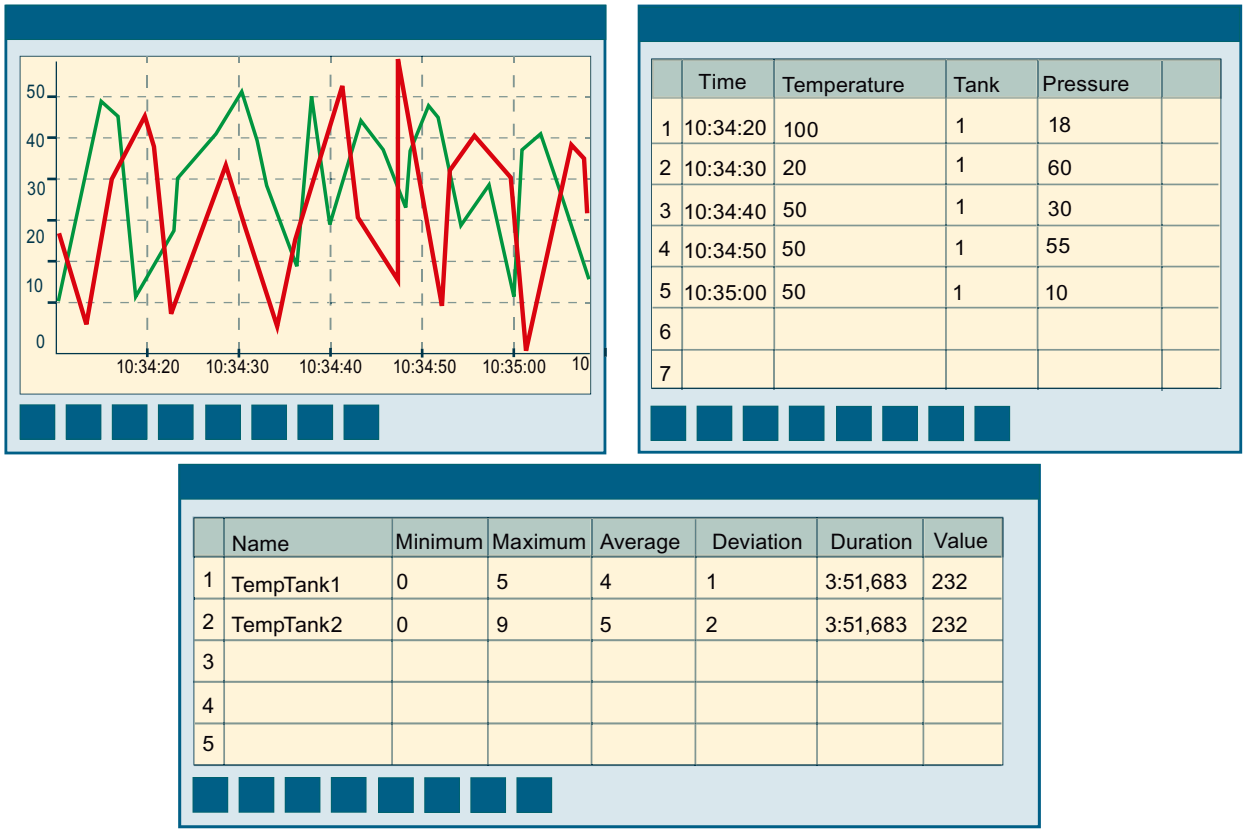
Controls for displaying tag values

To display tag values as a trend, use the trend control. The versions of trend views are available:

- "Trend control": You display a tag value over time, for example, the change in temperature. You can compare the current values and logged values or monitor the change in current values on the HMI device.
- "Function trend control": You display a tag value against a second tag value, for example, the engine speed against the heat produced.

You can use the "Trend companion" to create statistics, for example, from the displayed values. Furthermore, you can use the trend companion as reading assistance for the trend control.

To display tag values in a table, use the process control.



Displayed values

When configuring the trend control, specify which tag values are to be displayed.

- "Online": The trend is continued with current individual values from the PLC.
- "Log": In runtime, the trend control displays the values of a tag from a data log. The trend shows the logged values in a particular window in time. The operator can move the time window in runtime to view the desired information from the log.

Data types

Tags of the data type "Word" or "Int" and array tags of the data type "Word" or "Int" are permitted for display in a curve.

5.5.1.2 Outputting tag values as trends

Introduction

You have the option of displaying the values of tags graphically in runtime with the help of the following controls:

- You can visualize the trend control to display currently pending process values or logged values in runtime as trends over time.
- You use the function trend control to visualize currently pending process values or logged values in runtime as trends in relation to other tags.

The axis designations are different for the two trend views:

- The trend control has a "Time axis" and a "Value axis"
- The function trend control has an "X axis" and a "Y axis"

You can display up to nine trends in both the function trend control and the trend control.

Structure of a trend control

Configure the trend control appearance in the Inspector window: You define the number of trend areas and configure the trends it contains.

You can configure multiple trends, value axes and time axes for each trend area. You can alter the appearance, labeling and assignment in the Inspector window for each individual trend, value axis and time axis created.

Configure trend areas

You can divide the display area into multiple trend areas, if necessary. Each area functions like a standalone trend control. This allows you to show temperature changes or values from different days, for example, as trends and compare them. The "Range proportion" specifies how much space is provided for a given area in the trend control.

Each range proportion is calculated on the basis of the total number of range components. If you have configured a total of three trend areas, for example, a range proportion of "1" will

result in three trend areas of equal size. To increase the range proportions in relation to each other, increase the range proportion of one or more trend areas.

Configuring axes

You configure the axes of the trend control for each trend area in the properties of the trend areas.

The following properties are set by default with a value axis:

- The value range is based on the current values of the assigned trend
- The value axis scale is linear and based on the value range
  Alternatively, you can configure a logarithmic scale:
  - In logarithmic scaling, only positive values are displayed.
  - In negative logarithmic scaling, only negative values are displayed.

If you configure a value axis for the trend control, you can also set up axis segments. You assign a value range and a display name to each axis segment.

In the function trend control, the value axis corresponds to both the "X axis" and the "Y axis".

Configuring trends

You configure the axes for each trend area:

- The time and value axes in the trend control
- The different value axes in the function trend control

By default, the data area is based on the current values of the associated trend.

You can also configure the visualization of limits and values with "Uncertain status" for a trend. If a trend exceeds or falls below a configured limit, the trend is colored.

Configuring the time axis and time range (trend control)

The time range for trend display is configured with time axes. In a trend control, you can create multiple time axes that you can assign to one or more trend areas.

If you configure several time axes to a trend control, the sequence of the time axes in the Inspector window corresponds to the sequence in the trend control. If multiple time axes run along the same side of a trend control, the first time axis in the list is at the bottom left. The last time axis is at the top right.

5.5.1.3 Representing multiple trends

Introduction

If you display several trends simultaneously in the trend control, assign each trend its own value and time axis. Alternatively, you can assign a shared time and/or value axis to several trends.

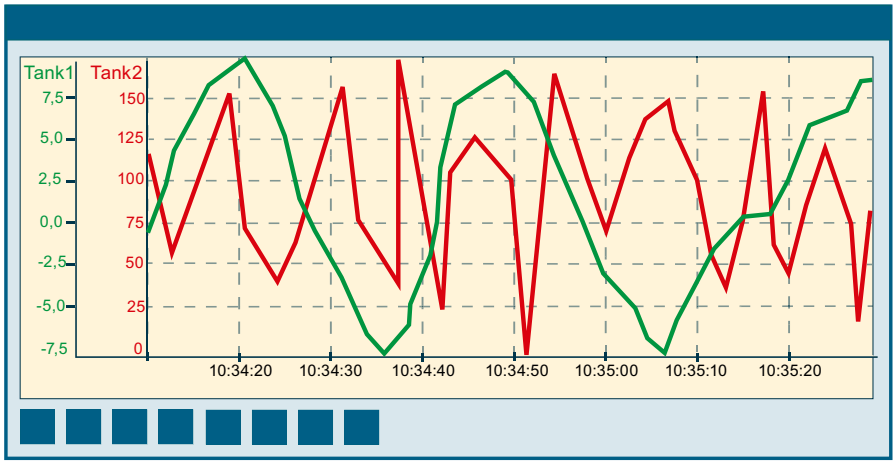
You configure the axes of a trend control for each individual trend in the Inspector window under "Properties > Trend areas".

The axes are assigned to the configured trends in the Inspector window under "Properties > Trend areas > Trends".

Representation using different axes

If the values to be displayed in a trend control differ greatly, a common value axis makes no sense. If you assign each trend its own value axis, they should also display different scales. Individual axes can be hidden if required.

The figure below shows two trends with different value axes using a trend control as an example:

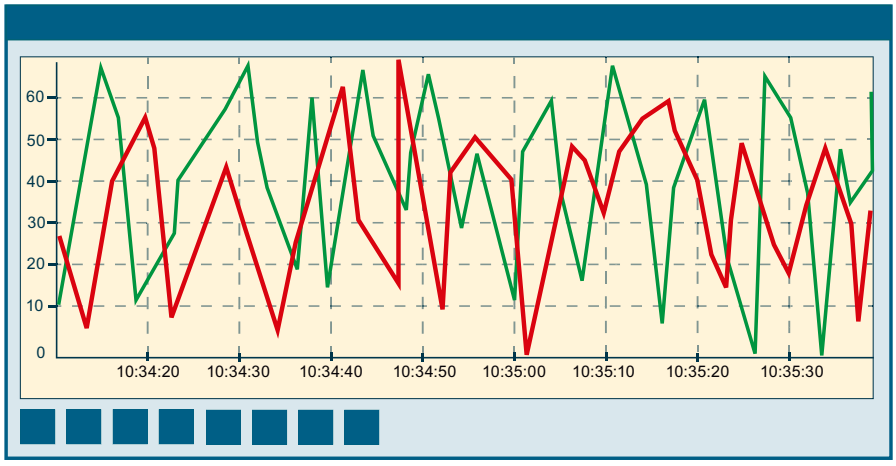


Representation using common axes

If the comparability of the trend directions is important, common axes in a trend control is sensible. Connected trend views can have a common time axis.

If you configure trends with a shared time axis, use tags with the same update cycle for the data supply.

In the case of different updating cycles, the length of the time axis is not identical for all tags. Since the trends are updated at different times due to the different updating cycles, a minimal different in the end time for the time axis occurs on each change. As a result, the trends displayed skip slightly to and fro on each change.



5.5.1.4 Basics of time range

Time range

The time range is the range from which the values at the HMI device are shown. The time range is determined by the start time and the end time. The time range is always in the past. If the end time is later than the current system time, the current system time is used as a temporary end time.

A distinction is made between the following time ranges:

- Static time range
- Dynamic time range

Static time range

The static time range is determined by fixed start and end times. The values are displayed within this time range.

Dynamic time range

The dynamic time range is determined by a period of time beginning with a fixed start time. The end time thus corresponds to the conclusion of the time period.

You set the time period as follows:

- Duration, e.g. 30 minutes
- The number of measurement points multiplied by the update cycle also produces a duration.

Configuring time range

Configure the time range for all controls. Configure the time range in the time column or in the time axis for the process control and the trend control. For the function trend control, configure the time range directly at the trend.

You select one of three options for the time range:

- "Time span": You define the time range using a starting time and a following time span. You set the time interval with the settings "Time range base" and "Time range factor", for example, 30 minutes.
- "Start time and end time": You define the time range using a starting time and an end time.
- "Measuring points": You define the time range using a starting time and a number of measuring points.

5.5.1.5 Representing trend directions

Introduction

In a trend control, you display a trend direction with one of the following modes:

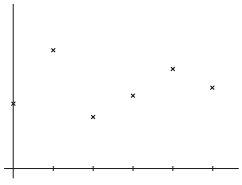
- Dots
- Interpolated
- Stepped
- Values

Select "Properties > Trend areas > Trends > Trend mode" to configure the trend display in the Inspector window.

Dots

Values are shown as dots. The display of the points can be configured as you wish.

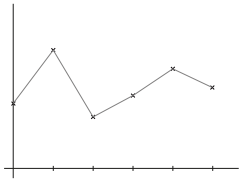
The following image shows the trend direction with the format pattern "Dots":



Interpolated

The trend curve is interpolated on a linear basis from the values. The display of the lines and points can be configured as you wish.

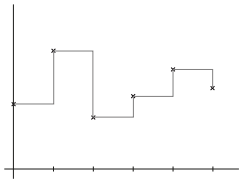
The following image shows the trend direction with the format pattern "Interpolated":



Stepped

The trend curve is interpolated as a stepped curve from the values. The display of the lines and points can be configured as you wish.

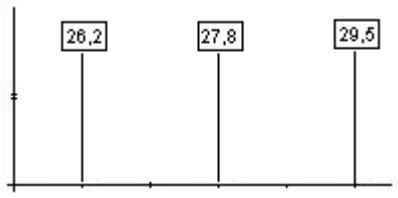
The following image shows the trend direction with the format pattern "Stepped":



Values

The values are displayed as text at each time stamp or each main grid line of the time axis. A unit can be displayed additionally for the values.

The figure below shows the trend direction with the format pattern "Values":



5.5.1.6 Outputting tag values in tabular format

Introduction

To display tag values in tables in runtime, add a process control to a screen. A time stamp is displayed for each value. The values are displayed in value columns, and the time stamps in time columns. Assign the time column to one or several value columns. You have the option of configuring a time column and nine value columns in the process control.

If you assign multiple value and/or time columns to a process control, the sequence of columns in the Inspector window corresponds to the sequence in the process control. If you assign the same time column to multiple value columns, the value columns in the list are automatically grouped according to the assigned time column.

- The time range for the table display is configured using the time column. A table has a common time column for multiple value columns. The first column [0] in the process view of the process control is the time column.
- You configure the values of the process control using value columns. You can display several value columns in a table, for example to compare the fill levels of several containers. Each value column is connected to the time column.

Configuration options in the process control

You can configure the following properties in the process control in line with your requirements:

- Configure the appearance of the process control:
  - Colors
  - Time base
  - Window settings of the control
- Configure the columns of the process control in the Inspector window.
  - Configure the time range using the time columns. A table can have a common time column for multiple value columns as well as separate time columns.
  - Configure the display of the tag values using the value columns. Each value column is connected to a time column. The value columns can have a common time column.
- Configure the appearance of the table
- Configure the toolbar and information bar of the process control.
- If required, configure data export from the process control.

5.5.1.7 Configuring tag evaluation

Introduction

Also configure a trend companion if you want to evaluate data from the trend control in runtime. You can also configure the trend companion as "Ruler".

You connect the trend companion to one of the following controls:

- Trend control
- Function trend control

Set a "Display mode" in the trend companion. The display mode determines which data is shown in the trend companion.

The contents of the trend companion are shown in columns. The available columns depend on the connected control. A block is assigned to each column. You define the alignment and appearance of the column headers using the blocks. By default, the format of the connected control, for example the time display, is used for the display format.

Configuration options in the trend companion

You can configure the following properties in the trend companion in line with your requirements:

- Configure the view of the trend companion in the Inspector window:
- Select the "Mode" of the trend companion under "Properties > General".
- Configure the display, labeling and sequence of the columns.

Display modes

Three different display modes are available in the trend companion:

- Ruler mode
  The ruler window shows the coordinate values of the trends on the ruler or values of a selected row in the table.
- Statistics area mode
  The statistics area window shows the values of the lower limit and upper limit of the trends between two rulers or the selected area in the table. You can only connect the statistics area window to the trend control or the process control.
- Statistics mode
  The statistics window displays the statistical evaluation of the trends. The statistics include:
  - Minimum
  - Maximum
  - Average
  - Standard deviation
  - Integral

All windows can also display additional information on the connected trends or columns, such as time stamps.

5.5.2 Configuring a trend control

Introduction

For the graphic display of tag values in runtime, add a trend control to a screen. The trend control allows you to display current and logged values for a specific time window, for example. For the display of data logs in runtime, you can move the time window to view the logged values.

The list of elements in a group always starts with 0, for example trend [0] is the first trend that has already been created in the object. For a clearer display of multiple trends, you can configure different names, line types and colors.

Requirement

- Data log with backup has been configured.
- The HMI tag for temperature measurement has been configured, for example "MotorTemperature".
- The HMI tag for velocity measurement has been configured, for example "MotorSpeed".
- A screen has been configured.

Configuring the trend area and axes

1. Add the "trend control" object to the screen from the "Toolbox" task card.
2. Go to "Properties" and set the required height, width and position of the object.
3. Open the "Trend areas" group under "Properties".
   The index numbers of the trend areas created for the object are displayed.
4. Expand the index number of the first trend area.
   The properties of the first trend area are displayed.

   Note

   To add another trend area, go to "Properties > Trend areas > [0] trend areas > Trends" and click the selection button in the "Static value" column. In the dialog, click "Add".

5. Define the colors for displaying the trend area and the reference lines.
6. Configure the time axis and value axis settings under "Bottom time axis" and "Left value axis".

Configuring trends

1. Go to "Trend areas > [0] trend areas > Trends" and click on the selection button in the "Static value" column.
   A dialog opens.
2. Click "Add" in the "Index" column.
   This adds another trend. Close the dialog.
3. Expand the index number of the first trend [0]. The trend settings are displayed.
4. Specify the name of the first trend under "Display name", for example "Speed".
5. Select the entry "Online" under "Data source Y > Source".
6. Under "Tag" enter the tag "MotorSpeed".
7. Configure the line color for the trend, for example, blue.
8. Expand the index number of the second trend [1]. The trend settings are displayed.
9. Specify the name of the second trend under "Display name", for example "Temperature".
10. Specify "Online" as the source type under "Data source" and enter the name of the tag "MotorTemperature".
11. Configure the line color for the trend, for example, red.

Result

The trend control is now configured. In runtime, you monitor value changes over time on the basis of two trends. One trend shows the temperatures measured and the other trend the velocity.

Configure an additional value display if you want to evaluate the data of the trend control in runtime. You can also configure the value display as a "Ruler".

See also

Trend control (Page 315)

Configuring trend control for plant objects (Page 7085)

5.5.3 Configuring the function trend control

Introduction

You use the function trend control to represent the values of a tag as a function of another tag. This means that you can present temperature trends as a function of the velocity, for example.

You can also compare the trend to a setpoint trend.

Requirement

- Data log with backup has been configured.
- The HMI tag for temperature measurement has been configured, for example "MotorTemperature".
- The HMI tag for velocity measurement has been configured, for example "MotorSpeed".
- A screen has been configured.

Configuring function trend areas and axes

1. Add the "trend control" object to the screen from the "Toolbox" task card.
2. Go to "Properties" and set the required height, width and position of the object.
3. Open the "Function trend area" group under "Properties".
   The index numbers of the function trend areas created for the object are displayed.
4. Expand the index number of the first function trend area.
   The properties of the first function trend area are displayed.

   Note

   To add another function trend area, go to "Properties > Function trend area > [0] function trend area > Function trends" and click on the selection button in the "Static value" column. In the dialog, click "Add".

5. Enter a meaningful name for the function trend area, for example, "SpeedToTemperature".
6. Open the temperature value axis settings under "Left value axis".
7. Define the value range for temperature, for example, by entering a maximum scale value of 350 degrees and a minimum scale value of 0 degrees.
8. Open the velocity value axis settings under "Bottom value axis".
9. Define the value range for speed, for example, by entering a maximum scale value of 1400 rpm and a minimum scale value of 0 rpm.

Note

Available scaling types

The f(x) trend view supports the "Linear" scaling type.

Configuring trends

1. Go to "Function trend area > [0] function trend area > Function trends > [0] function trend".
2. Specify "Online" as the source type under "Data source X", and enter the name of the process tag "MotorTemperature" under "Tag".
3. Specify "Online" as the source type under "Data source Y", and enter the name of the process tag "MotorSpeed" under "Tag".
4. Specify the time range of 1 second under "Properties > Function trend area > Function trends".

Result

The function trend control is now configured. In runtime, you monitor value changes on the basis of two trends. One trend shows the temperatures and the other trend the speed. In the function trend control, you can, for example, monitor how the temperature of the motor increases as the velocity increases.

Configure an additional value display if you want to evaluate the data of the trend control in runtime. You can also configure the value display as a "Ruler".

See also

Function trend control (Page 319)

5.5.4 Configuring bit-triggered trends

By setting a trigger bit in the "Trend transfer" tag, the HMI device either reads in a trend value or an entire trend buffer. This is defined in the configuration. Bit-triggered trends are usually used to represent fast changing values.

To trigger bit-triggered trends, appropriate external tags must be created in the "Tags" editor and connected to trend areas during configuration. The HMI device and PLC then communicate with each other via these trend areas.

Explanation of terms

Trend buffer

External array tag, the values of which are displayed as a trend. The number of array elements must be the same as the number of the measuring values of the trend. The tag is only available if the archive has not been selected as the "Source settings".

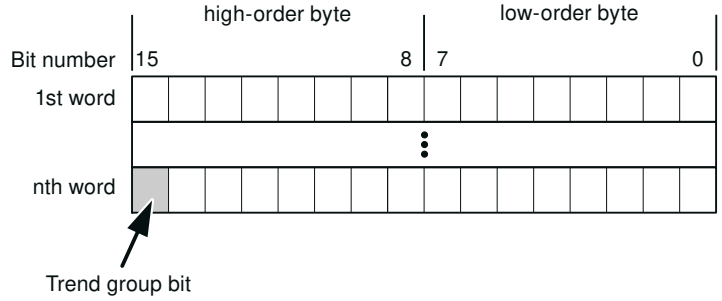
Trend request

When you open a screen with one or more trends on the HMI device, the HMI device sets the associated bits in the trend request area. After deselecting the screen, the HMI device resets the corresponding bits in the trend request area.

The trend request area can be used in the PLC to evaluate which trend is currently displayed on the HMI device. Trends can also be triggered without evaluation of the trend request area.

Trend transfer 1

This area is used to trigger trends. In your control program, set the bit assigned to the trend in the trend transfer area and the trend group bit. The trend group bit is the last bit in the trend transfer area. The HMI device detects the triggering and reads either a value or the entire buffer from the PLC. It then resets the trend bit and the trend group bit.



The trend transfer area must not be changed by the control program until the trend group bit has been reset.

Switch buffer

An external array tag with trend values as a second buffer that can be set when configuring a trend. While the HMI device reads the values from buffer 1, the PLC writes to buffer 2. If the HMI device reads buffer 2, the control writes to buffer 1. This prevents the trend values from being overwritten by the PLC while the HMI device is reading the trend.

Trend transfer 2 (required only with switch buffers)

Trend transfer 2 is required for trends that are configured with switch buffers. It is structured in the same way as trend transfer 1.

Trigger bit

Each trend is assigned a specific bit for communication between the HMI device and the PLC. If you assign trigger bit "4" to a trend, for example, the trend is identified by bit 4 in the trend request and in the trend transfers.
Do not use the group bit as trigger bit. The HMI device uses the group bit to detect the trigger signal. The position of the group bit in the trend transfer depends on the selected PLC.

Requirement

The following data blocks of the PLC were configured in STEP 7:

- Trend buffer (trend tag of the Array type)
- Trend request
- Trend transfer 1

A screen is configured.

Configuring the bit-triggered trend without a switch buffer

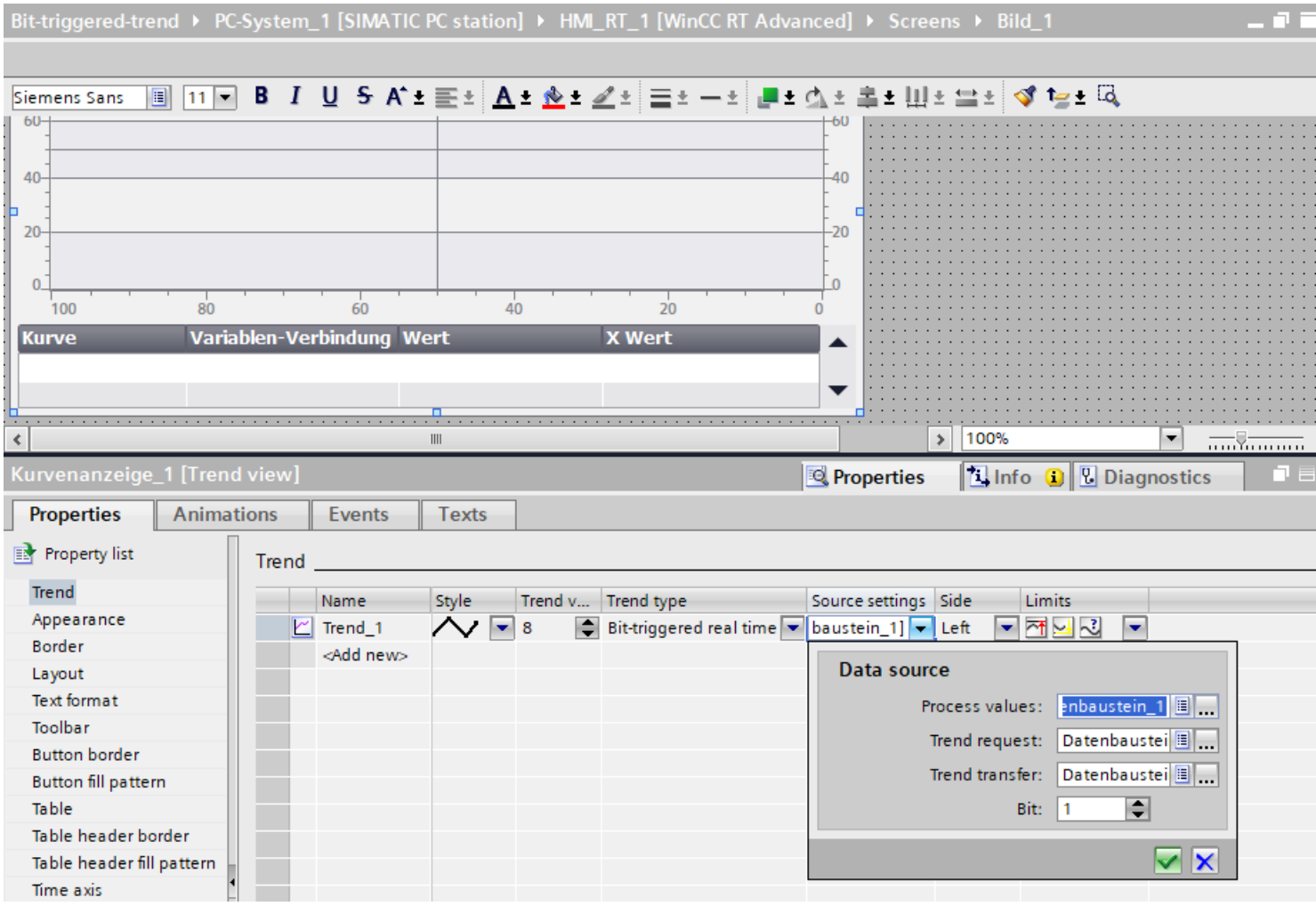
1. Add the "Trend control" object to the screen from the "Toolbox" task card.

2. Set the desired height, width and position for the object under "Properties > Layout" in the Inspector window.
3. Expand the "Time axis" group under "Properties" and select the "Points" axis mode under "Settings". Under "Range", specify how many values are to be displayed on the X axis. Under "Label", select the scale labeling of the intermediate values, the step size and the number of graduation lines.
4. Under "Properties", expand the "Trend" group.
5. In the "Name" column, click the "Add new" entry and specify the name for the trend. If needed, change the style of the trend (color, line, mode).
6. In the "Trend values" column, specify the number of values. This number depends on how many array tags you have assigned for the "Trend buffers".

7. In the "Trend type" column, select the "Bit-triggered real time" variant.
8. Set in the column "Source settings > Data source" the tags for:
   - Process values
   - Trend request
   - Trend transfer

   Then enter the bit position in "Trend request" and "Trend transfer".

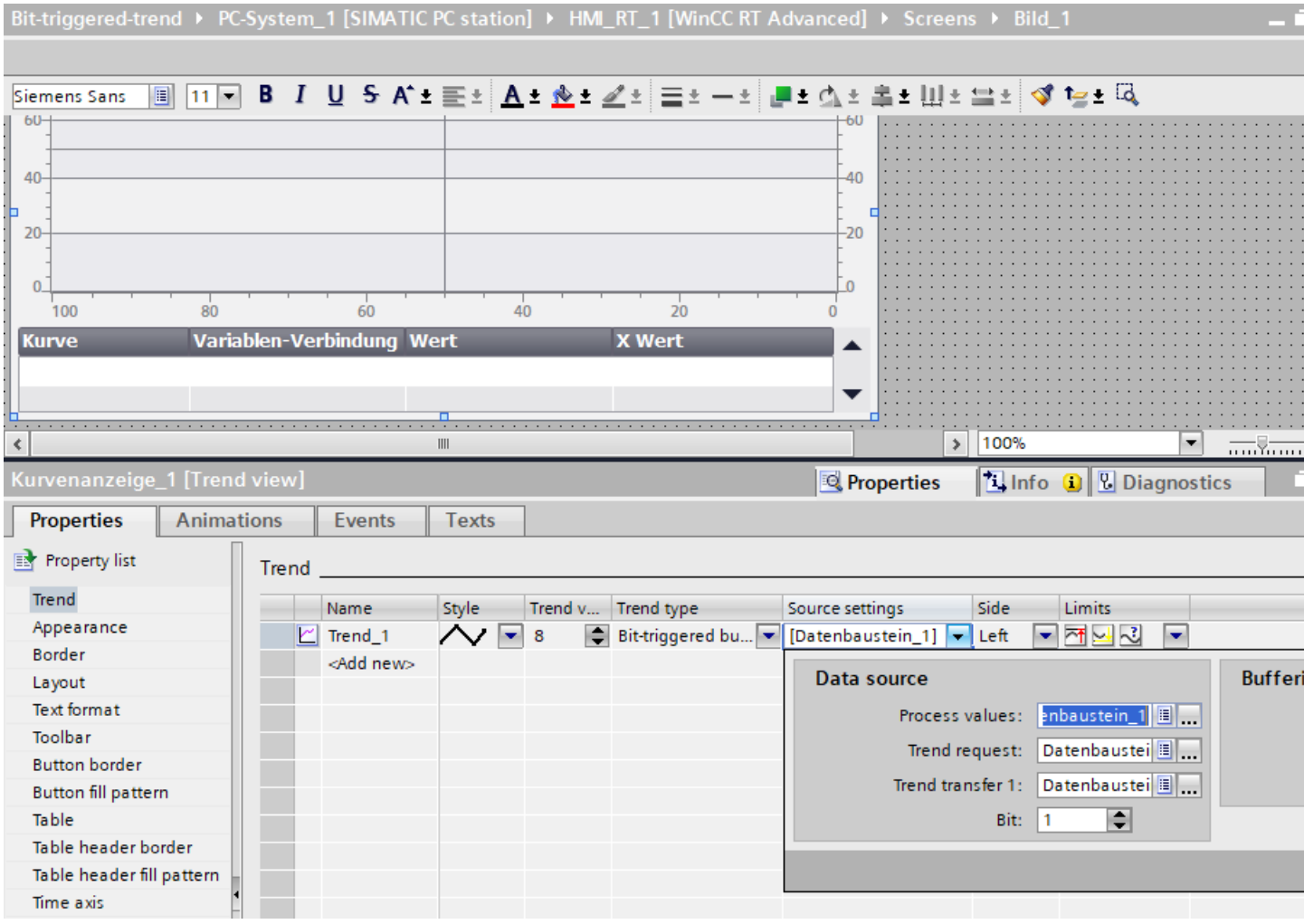


Configuring the bit-triggered trend with switch buffer

The procedure is first the same as before for the configuration of a bit-triggered trend without switch buffer (steps 1-6).

1. Select the variant "Bit-triggered buffer" in the "Trend type" column.
2. Select in the column "Source settings > Data source" the tags for:
   - Process values
   - Trend request
   - Trend transfer 1

   Then enter the bit position in "Trend request" and "Trend transfer".
3. Check the "Activate buffer" check box and select the tags for:
   - Buffer tag
   - Trend transfer 2



5.5.5 Configuring the process control

Introduction

To display tag values in tables in runtime, add a process control to a screen. The time column shows the time at which the value was reached. The value columns show the values at a given time stamp.

You can use the process control to display the incoming temperature values of a motor in a table in runtime, for example.

Requirement

- HMI tag for temperature measurement has been configured, for example "MotorTemperature"
- Cycle for regular display updates has been configured
- The screen is open
- The Inspector window is open

Configuring the process control

1. Add the required process control to the screen from the "Tools" task card.
2. Enter the label for the process control, for example "MotorTemperatureView", under "Properties" in the Inspector window.
3. Go to "Properties > Process view > Columns > [0]" and configure the time column with the time ranges for the table.
4. Under "Sort order", define the order in which the columns of the process control are shown.
5. Configure the "Time range" and "Format" of the time display in the time column, for example "Time span".
6. Set the start time, the basis and the factor for the time range, for example 10 minutes.
7. If the values in the time column are to be updated automatically, enable "Update".
8. Go to "Properties > Process view > Columns > [1]" and configure the properties for the value column.
9. Enter the name of the column, for example "Temperature".
10. Configure the type "Online" for current values under "Data source" and enter the tag "MotorTemperature" under "Tag".
11. Configure the display of content and the headers for the given value column.
12. Configure the toolbar and information bar of the process control.
13. If required, configure the security settings of the process control.

Result

The process control is now configured and displays the temperature of the motor at the measured time in runtime.

See also

Process control (Page 335)

Configuring reordering of the columns (Page 392)

5.5.6 Configuring the trend companion

Introduction

The trend companion allows you to display statistical data, for example mean values for the trend control with temperature trends. The calculation of statistical data gives the user access to trends and value changes over time. As well as calculating statistical data, you can use the trend companion as a viewing aid for trend values at a ruler position.

Requirement

- Trend control or function trend control has been configured in the screen
- Cycle for regular display updates has been configured
- The screen is open
- The Inspector window is open

Procedure

1. Add the required trend companion to the screen from the "Toolbox" task card.
2. Select the relevant control under "Properties > Data source" in the Inspector window to connect the trend companion to the selected control.
3. To display the trend companion below the selected control, select the option "Dock to data source".
4. Select the "Trend companion mode" of the trend companion under "Properties", for example "Statistic result".
5. Configure the appearance of the selected mode under "Properties > Statistic mode appearance":
   - Change the colors, row height and fonts in the trend companion if required.
   - Configure the headers under "Properties > Statistic mode appearance > Header settings" if required.

6. Configure the trend companion columns under "Properties > Statistic mode appearance > Columns":
   – Change the display, labels or order of columns if required.
7. Configure the view of the trend companion in the Inspector window:
   – Change the display, labeling and colors of the trend companion if required, or use the colors of the control to which the trend companion is docked.
   – Configure the information bar and the toolbar of the trend companion.

   Note

   The "Print" button [3] is reserved for future versions

Configuring selection

If, for example, a user wants to export values from a row, they must select the row. You specify the selection range and colors for selection during configuration. You define the settings for selection for each display mode.

1. In the Inspector window, go to "Properties > Trend ruler appearance" and select the "Selection mode" for the selection range, for example "Multiple elements".
2. Select the color mode for selection, for example rows.
3. If required, select the "Border color" and "Border width" to be displayed around the selection area.
4. Choose the colors for selection as required.

Result

The trend companion is configured. The statistical values calculated are displayed in the trend companion in runtime.

See also

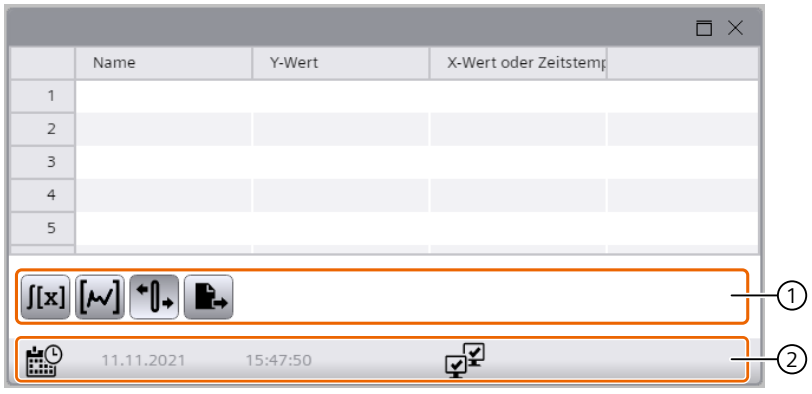
Trend companion (Page 323)

Configuring reordering of the columns (Page 392)

5.5.7 Configuring the toolbar and information bar

Introduction

You can operate the controls in runtime using the buttons in the toolbar. The information bar displays the status messages of the control. During configuration, you define the contents of the toolbar and information bar.



① Toolbar
② Information bar

Requirement

- You have opened the screen which contains at least one object, for example, the trend companion.
- The Inspector window is open.

Configuring the toolbar

To configure the toolbar, follow these steps:

1. In the Inspector window under "Properties > Miscellaneous > Toolbar", configure the general properties of the toolbar, such as background color or visibility.
2. In the Inspector window, under "Properties > Properties > Miscellaneous > Toolbar > Elements > Button > Visibility", enable the buttons that you need in Runtime.
3. Configure the button display, for example, background color, border and size.
4. Under "Properties > Properties > Miscellaneous > Toolbar > Elements > Button > Authorization", select the authorization that is required in Runtime to operate the button.
5. When a button is not operated in Runtime, disable "Allow operator control". You can reactivate a disabled a button by using a script in runtime, for example.

Configuring the information bar

To configure the information bar, follow these steps:

1. In the Inspector window under "Properties > Properties > Miscellaneous > Information bar", configure the general properties of the information bar, such as background color or visibility.
2. In the Inspector window under "Properties > Properties > Miscellaneous > Information bar > Elements > State display > Visibility", enable the elements that you need in Runtime.
3. Configure the display of the respective element.
4. Select the authorization that is required in Runtime to operate the element.
5. When an element is not operated in Runtime, disable "Allow operator control". You can enable a disabled element again, for example, with a script in Runtime.

5.5.8 Defining the data source

Introduction

Using the data source, you define the sources from which the values are displayed on the HMI device in runtime. The following sources are available:

- Current process values from tags
- Archived values from logging tags

To set up data supply for the controls over a tag, enter the name of the tag in the "Static value" column under "Data source > Tag".

Requirement

- An online tag or logging tag is configured
- Value column or trend has been created
- The Inspector window is open

Displaying current process values

Proceed as follows to display current process values:

1. Click "Properties > Process view > Columns" in the Inspector window to define the data source for a process control.
   The first column is always reserved for the time column. You enter the data source for value columns [1] to [N].
2. Click "Properties > Trend areas > Trends" in the Inspector window to define the data source for a trend control.
   For the function trend control, click on "Properties > Function trend area > Function trends".
3. Configure the "data source":
   - Select the entry "Online" as "Source type".
   - When you configure the trend of an function trend control, enter one tag each for "Data source X" and "Data source Y".
   - When you configure the trend of a trend control or a value column, enter the corresponding tag under "Tag".
   - Select the update cycle.

Note

Using UDTs

In the "Static value" column under "Tag" first enter the name of the data type and then the name of the element separated by a period, for example, "PLCDatatypeName.ElementName".

Displaying values from a log

Proceed as follows to display values from a log:

1. Click "Properties > Process view > Columns" in the Inspector window to define the data source for a process control.
   The first column is always reserved for the time column. You enter the data source for value columns [1] to [N].
2. Click "Properties > Trend areas > Trends" in the Inspector window to define the data source for a trend control.
   For the function trend control, click on "Properties > Function trend area > Function trends".
3. Configure the "data source":
   - Select the entry "Logs" as "Source".
   - When you configure the trend of an function trend control, enter one tag each for "Data source X" and "Data source Y".
   - When you configure the trend of a trend control or a value column, enter the corresponding tag under "Tag".

Note

Using logging tags

In the "Static value" column under "Tag" first enter the name of the HMI tag and then the name of the associated logging tag separated by a period, for example, "HMITag_1:LoggingTag_1".

5.6 Reference

5.6.1 Quality codes of HMI tags

Introduction

The "Quality Code" is required to evaluate the status and quality of a tag. The quality of the entire value transfer and value processing of the respective HMI tag is summarized in the indicated Quality Code. For example, it is possible to determine from the Quality Code whether the current value is a start value or substitute value.

The quality codes are prioritized. If several codes occur at the same time, the Quality Code reflecting the lowest quality is displayed.

Evaluation of Quality Codes

You can evaluate the Quality Code in a number of different ways:

- Evaluation with JScript functions
- Evaluation using the "Quality Code changed" event of an I/O field.

Structure

The Quality Code has the following binary structure:

High byte: Specific information for WinCC Unified								Low byte: Quality code according to PROFIBUS PA or OPC DA							
Flags				Enhanced substatus				Quality		Substatus				Limits	
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1

Quality (bit 7 and bit 8)

Quality represents the basic values of the quality levels. Making use of the substatus and limits gives rise to intermediate values over and above the quality stage concerned.

Bit 8	Bit 7	
0	0	Bad - The value is not useful. The reasons are indicated by the sub-status.
0	1	Uncertain - The quality of the value is less than normal, but the value may still be useful. The reasons are indicated by the sub-status.
1	0	Good (Non-Cascade) - The quality of the value is good. Possible alarm conditions may be indicated by the sub-status.
1	1	Good (Cascade) - The quality of the value is good and may be used in control.

Flags (bit 12 to bit 15)

Flags contain information on the interpretation of the Quality Code.

Bit 12	Source quality	0: The data quality has been determined and assigned by external data source.
Bit 13	Source time	1: The data timestamp has been produced and assigned by external data source.
Bit 14	Time corrected	1: The data timestamp applied by external data source has been corrected by the system. Thus, Bit 13 "Source time" is not set. Time correction happens if the external timestamp is older than the timestamp of the last known value.
Bit 15	reserved	

Sub-status and extended sub-status

The quality alone is not enough. Substatuses divide the individual qualities. The Quality Code is binary-coded. The value must be converted to hexadecimal format for the analysis of the Quality Code.

Externally generated quality code of tags

If bit 12 is not set, the Quality Code was generated from an external source in accordance with PROFIBUS PA. The table begins with the worst Quality Code and ends with the best Quality Code. The best Quality Code has the lowest priority, while the worst Quality Code has the highest priority. If several statuses occur for one tag in the process, the poorest code is passed on.

Code (hex)	Quality		Q	Q	S	S	S	S	L	L
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3C	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-

Code (hex)	Quality		Q	Q	S	S	S	S	L	L
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x00	Bad	non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain. Used for propagation.	0	1	0	0	0	0	-	-
0xE0	Good(Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	0	1	1	0	-	-

Code (hex)	Quality		Q	Q	S	S	S	S	L	L
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0xC0	Good(Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Internally generated quality code of tags

If bit 12 is set, the Quality Code was generated from the HMI system. The table begins with the worst Quality Code and ends with the best Quality Code. The best Quality Code has the lowest priority, while the worst Quality Code has the highest priority. If several statuses occur for one tag in the process, the poorest code is passed on.

Code (Hex)	Quality		Q	Q	S	S	S	S	L	L
0x70n	Bad	Disabled	0	0	0	0	-	-	-	-
0x300	Bad	Unusable value - A logged value has been identified to be incorrect, but a respective correction value is not available. The corresponding sub-status is set to 'non-specific'.	0	0	0	0	0	0	0	0
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3F	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x100	Bad	Aggregated value - The value has been calculated out of multiple values with less than the re-quired number of good sources. This includes data aggregation by means of data compression algorithms. The corresponding sub-status is set to 'non-specific'.	0	0	0	0	0	0	0	0
0x00	Bad	non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x74n	Uncertain	Disabled - The provider of the value, e.g. logging tag for logged value, has been disabled and the previous value was GOOD or UNCERTAIN. In case of GOOD the corresponding sub- status is set to 'last usable value'. In case of UNCERTAIN the corresponding sub-status is taken from the last sub-status.	0	1	0	0	-	-	-	-

Code (Hex)	Quality		Q	Q	S	S	S	S	L	L
0x158	Uncertain	Aggregated value - The value has been calculated out of multiple values with less than the required number of good sources to be GOOD as well as less than required number of bad sources to be BAD. This includes data aggregation by means of data compression algorithms. The corresponding sub-status is set to 'sub-normal'.	0	1	0	1	1	0	0	0
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain. Used for propagation.	0	1	0	0	0	0	-	-
0x3C0	Good(Cascade)	Corrected value - A logged value has been corrected. The corresponding sub-status is set to 'non-specific'.	1	1	0	0	0	0	0	0
0x2C0	Good(Cascade)	Manual input - A logged value has been created manually. The corresponding sub-status is set to 'non-specific'.	1	1	0	0	0	0	0	0
0x1C0	Good(Cascade)	Aggregated value -The value has been calculated out of multiple (GOOD) values. This includes data aggregation by means of data compression algorithms. The corresponding sub-status is set to 'non-specific'.	1	1	0	0	0	0	0	0
0xE0	Good (Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	1	0	0	0	-	-
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-

Code (Hex)	Quality		Q	Q	S	S	S	S	L	L
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0x6C0	Good(Cascade)	Initial value - The local data source has been initialized with the configured initial value. The corresponding sub-status is set to 'non-specific'.	1	1	0	0	0	0	0	0
0c4C0	Good(Cascade)	Last usable value - The local data source has been initialized with the last usable value, if pre-sent inside a local persistency. The corresponding sub-status is set to 'non-specific'.	1	1	0	0	0	0	0	0
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0xC0	Good(Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Limit

The Quality Codes can be further subdivided by limits. Limits are optional.

	Q	Q	S	S	S	S	L	L
O.K. - The value is free to move.	-	-	-	-	-	-	0	0
Low limited - The value has acceded its low limits.	-	-	-	-	-	-	0	1
High limited - The value has acceded its high limits.	-	-	-	-	-	-	1	0
Constant (high and low limited) - The value cannot move, no matter what the process does.	-	-	-	-	-	-	1	1

Quality Codes in Communication with OPC

When connecting to a OPC UA server, the OPC UA status code is shown in a quality code.

Quality Code in WinCC	Status code according to OPC
0x48	0x40
0x4C	0x40
0x5C	0x40
0x60	0x40
0x80...0xD4	0xC0
0xD8	0xC0

5.6.2 Data types

5.6.2.1 Data types for SIMATIC S7-300/400

Overview

The following table shows the data types for SIMATIC S7-300/400 with the corresponding HMI data types and value ranges in WinCC.

Data type	Value range
Array of Char	• Array of Char is mapped in HMI to a tag of the String data type • The maximum array of Char is 254 characters. • With SIMATIC S7-300 – Read access: 222 characters – Write access: 212 characters • 0 ... 224 (ASCII) • Characters from the Windows 1252 code page
Bool	0 (FALSE), 1 (TRUE)
Byte	0 ... 255
Char	0 ... 255 (ASCII)

Data type	Value range
Counter	0 ... 999
Date	1990-01-01 ... 2168-12-31
DInt	–2147483648 ... +2147483647
DWord	0 .. 4294967295
Int	–32768 ... 32767
Real	±1.17549E-38 to ±3.40282E+38 and 0.0
S5Time	0 ... 2h46m30s0ms
String	• The maximum length of a string is 254 characters. • With SIMATIC S7-300 – Read access: 220 characters – Write access: 210 characters • ASCII • Characters from the Windows 1252 code page
Time [1]	-24d20h31m23s648ms ... +24d20h31m23s647ms
Time_Of_Day	00:00:00 ... 23:59:59.999
Timer	-0ms ... 2h46m30s0ms
Word	0 ... 65535

1: If the value is set via the HMI, then the granularity is in 100 nanosecond intervals. In contrast, the granularity of WinCC Advanced, WinCC Comfort and WinCC Professional is milliseconds.

5.6.2.2 Data types for SIMATIC S7-1200

Overview

The following table shows the data types for SIMATIC S7-1200 with the corresponding HMI data types and value ranges in WinCC.

Data type	Value range
Array of Char	• Array of Char is mapped in HMI to a tag of the String data type • The maximum array of Char is 254 characters. • 0 ... 255 (ASCII) • Characters from the Windows 1252 code page
Array of WChar	• Array of WChar is mapped in HMI to a tag of the String data type • The maximum array of WChar is 255 characters. • Characters from the Unicode code page
Bool	0 (FALSE), 1 (TRUE)
Byte	0 ... 255
Char	0 ... 255 (ASCII)
Date	1990-01-01 ... 2168-12-31
DInt	–2147483648 ... +2147483647
DTL	1970-01-01-00:00:00.0 ... 2262-04-11-23:47:16.854775807

Data type	Value range
DWord	0 ... 4294967295
Int	-32768 ... +32767
LReal	±1.79769313486231E+308 ... ±2.22507385850720E-308 and 0.0
Real	±1.17549E-38 ... ±-3.40282E+38 and 0.0
SInt	-128 ... +127
String	• The maximum length of a string is 254 characters. • ASCII • Characters from the Windows 1252 code page
Time [1]	-24d20h31m23s648ms ... +24d20h31m23s647ms
Time_Of_Day, TOD	00:00:00 ... 23:59:59.999
UDInt	0 ... 4294967295
UInt	0 ... 65535
USInt	0 ... 255
WChar	UNICODE
Word	0 ... 65535
WString	UNICODE
PLCUDT	-

1: If the value is set via the HMI, then the granularity is in 100 nanosecond intervals. In contrast, the granularity of WinCC Advanced, WinCC Comfort and WinCC Professional is milliseconds.

5.6.2.3 Data types for SIMATIC S7-1500

Overview

The following table shows the data types for SIMATIC S7-1500 with the corresponding HMI data types and value ranges in WinCC.

Data type	Value range
Array of Char	• Array of Char is mapped in HMI to a tag of the String data type • The maximum array of Char is 254 characters. • 0 ... 255 (ASCII) • Characters from the Windows 1252 code page
Array of WChar	• Array of WChar is mapped in HMI to a tag of the String data type • The maximum array of WChar is 255 characters. • Characters from the Unicode code page
Bool	0 (FALSE), 1 (TRUE)
Byte	0 ... 255
Char	0 ... 255 (ASCII)
Counter	0 ... 65535
Date	1990-01-01 ... 2168-12-31
Date_And_Time, DT	1990-1-1-0:0:0.0 ... 2089-12-31-23:59:59.999

Data type	Value range
DInt	–2147483648 ... +2147483647
DTL	1970-01-01-00:00:00.0 ... 2262-04-11-23:47:16.854775807
DWord	0 ... 4294967295
Int	-32768 ... +32767
LDT	1970-01-01-00:00:00.000000000 ... 2263-04-11-23:47:16.854775808
LInt	-9223372036854775808 ... +9223372036854775807
LReal	±1.79769313486231E+308 ... ±2.22507385850720E-308 and 0.0
LTime [1]	• Value range: -9223372036854775808 to 9223372036854775807 • Unit: ns • Resulting time interval: -106751d23h47m16s854ms775us808ns ... +106751d23h47m16s854ms775us807ns For more information on setting an LTime value via the HMI, refer to section IO field (Page 284).
LTime_Of_Day, LTOD	00:00:00.000000000 ... 23:59:59.999999999
LWord	0 ... 18446744073709551615
Real	±1.17549E-38 ... ±-3.40282E+38 and 0.0
S5Time	0ms ... 2h46m30s0ms
SInt	-128 ... +127
String	• The maximum length of a string is 254 characters. • ASCII • Characters from the Windows 1252 code page
Time [1]	-24d20h31m23s648ms ... +24d20h31m23s647ms
Time_Of_Day, TOD	00:00:00 ... 23:59:59.999
Timer	-24d20h31m23s648ms ... +24d20h31m23s647ms
UDInt	0 ... 4294967295
UInt	0 ... 65535
ULInt	0 ... 18446744073709551615
USInt	0 ... 255
WChar	UNICODE
Word	0 ... 65535
WString	UNICODE
PLCUDT	-

1: If the value is set via the HMI, then the granularity is in 100 nanosecond intervals. In contrast, the granularity of WinCC Advanced, WinCC Comfort and WinCC Professional is milliseconds.

5.6.2.4 User-defined PLC data types (UDT)

Overview

You can connect with the HMI tags and DB instances of user-defined PLC data types (UDT).

The PLC data type and the corresponding DB instances are created and updated centrally in STEP 7. In WinCC, you can use the following sources as PLC tag (DB instances):

- Data block elements that use a UDT as data type
- Instance data blocks of a UDT

The data type is taken from STEP 7 and is not converted into an HMI data type. The access type is always "Symbolic access".

Elements of a PLC data type

You have access to the following elements in WinCC with a structured PLC data type:

- Elements that have been released for WinCC in STEP 7.
- Elements whose data types are supported in WinCC.

Note

Invalid elements of a PLC data type in WinCC

Invalid elements generate an error in WinCC.

If you disable the "Accessible from HMI" option for the corresponding elements of the associated PLC data type in STEP 7, these elements are excluded in WinCC.

Naming conventions

The following characters are invalid in the name of the PLC data type and generate an error in WinCC:

- Period: "."
- Square brackets: "[" and "]"

Properties

The properties of the PLC data type and its elements are adopted in WinCC. Depending on the data type used, the properties are read-only or can be written to in WinCC.

If you change the connection of the PLC data type in WinCC, all elements of the PLC data type are deleted and the properties of the newly connected PLC tag are used.

In WinCC, you have access to STEP 7 comments on elements of the PLC data type.

You have limited access to properties in WinCC for the following elements of PLC data types:

- Elements of the data type "Struct"
- PLC data type

- Multidimensional arrays
- Array of complex data types except "DTL"

Mapping of the data type "DTL"

If a PLC data type contains elements of the data type "DTL", these elements are mapped in WinCC without lower-level elements. The data type "DTL" turns into "DateTime" in WinCC.

Tags with elements of the "DTL" data type

Tags that use the "DTL" data type element by element can only be used as read-only with symbolic addressing, e.g. with SIMATIC S7 1500. With absolute addressing, write access is also possible.

Configuring alarms

6

6.1 Basics

6.1.1 Alarm system

Introduction

Alarms display events, operating states or faults that occur or predominate in your plant. You can use alarms for diagnostic purposes for fault rectification, for example, and they help you rapidly to identify the cause of a fault. You can adjust your processes through targeted intervention so that compliant products continue to be produced despite the fault, or the process is stabilized and the fault only causes a minimal loss of production.

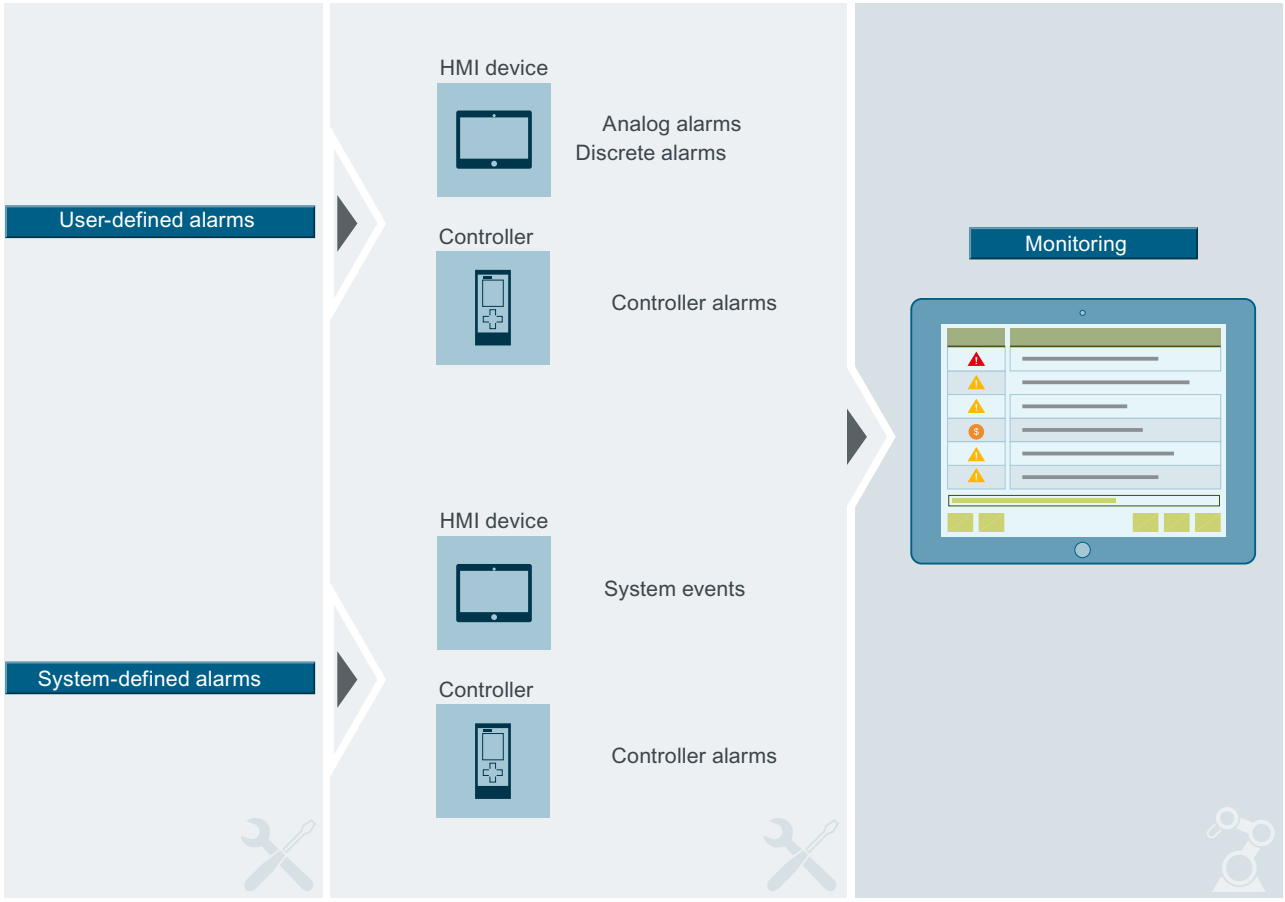
WinCC has a whole range of technical tools for implementing an alarm system. You use these tools to set up an alarm system that meets all requirements under currently applicable national and international standards and guidelines.

By means of the alarm system, events from the monitoring function in WinCC are displayed in form of alarms, acknowledged by the operator and logged, if necessary. To do this, alarms must be configured that are separated into alarm classes.

Alarm system

The alarm system distinguishes between the following alarms:

- User-defined alarms:
  - Analog alarms: Show limit violations (value changes), are used for monitoring the plant.
  - Discrete alarms: Show status changes, are used for monitoring the plant.
  - User-defined controller alarms: are configured in STEP 7, show status values of the controller, are used to monitor the plant.
- System-defined alarms:
  - System alarms: belong to the HMI device and are used to monitor it.
  - System-defined controller alarms: consist of system diagnostic alarms and system errors and are used to monitor the controller.



The detected alarm events are displayed on the HMI device.

Note

Note the following restrictions for controller alarms.

- WinCC only supports controller alarms of a SIMATIC S7-1500 controller.
- WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

Tips for an efficient procedure

- When you configure an alarm system, you need to take account of the abilities of future users.
- Alarm systems must be designed to use and allow for characteristic aspects of human perception.
- Important alarms must be highlighted so that they are noticed rapidly. The display of important information should be redundant to make it easier to see.
- Supplementary information about individual alarms ensures that faults are localized and cleared quickly.
- Information should if possible be directed at more than one sense (for example, visible and sound signals). Only alarm systems that meet these criteria will help the user to monitor and control the plant.

See also

Alarms (Page 709)
Alarm states (Page 714)
Alarm classes (Page 717)
Acknowledging alarms (Page 721)
Acknowledgment model (Page 715)
Alarm components and properties (Page 722)
Configuring analog alarms (Page 738)

6.1.2 Alarms

6.1.2.1 User-defined alarms

Analog Alarms

Description

Analog alarms indicate limit violations. You have defined in advance a limit value for the trigger tag and the trigger mode. An analog alarm is triggered depending on which mode you have defined, for example, when the value is higher than, lower than or the same as the defined value.

Example

The speed of a motor must not be too high or too low. You can configure analog alarms to monitor the speed of the motor. If the high or low limit for the speed of the motor is violated, an alarm is output on the HMI device containing the following alarm text, for example: "Motor speed is too low".

See also

Configuring analog alarms (Page 738)
Configuring optional parameters for discrete alarms and analog alarms (Page 745)
Discrete alarms (Page 710)
User-defined controller alarms (Page 710)
System events (Page 712)
System-defined controller alarms (Page 712)
Alarm system (Page 707)

Discrete alarms

Description

Discrete alarms indicate status changes in a plant. A discrete alarm is triggered when the value of a specific bit of an internal or external tag changes.

Example

Imagine that the state of a valve is to be monitored during operation. The two possible valve states are "opened" and "closed". In this case, a discrete alarm is configured for each valve state. A discrete alarm containing the following alarm text is output, for example, when the state of this valve changes: "Valve closed".

See also

Configuring discrete alarms (Page 734)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

Analog Alarms (Page 709)

User-defined controller alarms (Page 710)

System events (Page 712)

System-defined controller alarms (Page 712)

Alarm system (Page 707)

User-defined controller alarms

Example of an alarm

"The temperature in Tank 2 is too high."

Description

A user-defined controller alarm, e.g. a program alarm, created by the control project engineer in STEP 7. The PLC status values, such as time stamp and process values, are mapped in the controller alarm. If controller alarms are configured in STEP 7, accept them into the integrated WinCC operation as soon as a connection is established to the PLC. In STEP 7, the controller alarm is assigned to an alarm class. You import this alarm class with the controller alarm as a common alarm class.

Note

Automatic update of new or modified controller alarms on the HMI device

If controller alarms are configured in STEP 7 and an HMI connection to a SIMATIC S7-1500 controller (firmware version 2.0 or higher) is established, controller alarms are sent to the HMI device. After changes of the controller alarms, the HMI device configuration must no longer be transferred. The prerequisite is that the option "Central alarm management in the PLC" is enabled in the properties of the controller. In addition, the option "Automatic update" must be enabled in the runtime settings of the HMI device under "Alarms > Controller alarms".

Note

WinCC only supports controller alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

Controller alarms for HMI devices

If a controller is connected with one or more HMI devices, the configuration engineer assigns display classes to the controller alarms in STEP 7. The display classes determine the allocation to the HMI device. You activate the display classes that are to be available for each HMI device. In this case, only the controller alarms from this display class will be displayed on the HMI device. Up to 17 display classes are possible.

See also

Filtering controller alarms via display classes (Page 748)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Configuring automatic update of controller alarms on the HMI device (Page 797)

Analog Alarms (Page 709)

Discrete alarms (Page 710)

System events (Page 712)

System-defined controller alarms (Page 712)

Alarm system (Page 707)

6.1.2.2 System-defined alarms

System events

Example of an alarm

"Memory is full!"

Description

A system event indicates the system status and communication errors between the HMI device and system. System events are output in runtime in the configured alarm control. System events are output in the language currently set on your HMI device.

The time format (AM/PM or 24-hour format) is based on the selected language. If no translation of the alarm texts exists in this language, English is used as replacement and the corresponding time format is displayed.

See also

Editing system events (Page 748)

Analog Alarms (Page 709)

Discrete alarms (Page 710)

User-defined controller alarms (Page 710)

System-defined controller alarms (Page 712)

Alarm system (Page 707)

System-defined controller alarms

Example of an alarm

"CPU maintenance required"

Description

System-defined controller alarms are installed with STEP 7 and are only available if WinCC is operated in the STEP 7 environment.

System-defined controller alarms are used to monitor states and events of a controller. System-defined controller alarms consist of system diagnostic alarms and system errors (RSE)

Note

Automatic update of system diagnostic alarms on the HMI device

If an HMI connection to a SIMATIC S7-1500 controller (firmware version 2.0 or higher) is established, system diagnostic alarms are sent to the HMI device and automatically updated. The prerequisite is that the option "Central alarm management in the PLC" is enabled in the properties of the controller. In addition, the options "Automatic update" and "System diagnostics" must be enabled in the runtime settings of the HMI device under "Alarms > Controller alarms".

Note

Note the following restrictions:

- WinCC only supports system diagnostic alarms of a SIMATIC S7-1500 controller.
- WinCC only supports system diagnostic alarms that are automatically updated by the central alarm management in the controller.

See also

Configuring the display of system diagnostic alarms (Page 767)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Configuring automatic update of controller alarms on the HMI device (Page 797)

Analog Alarms (Page 709)

Discrete alarms (Page 710)

User-defined controller alarms (Page 710)

System events (Page 712)

Alarm system (Page 707)

6.1.3 Alarm states

Description

Every alarm has an alarm state. The alarm states are made up of the following events:

- **Incoming**
  The condition for triggering an alarm is fulfilled. The alarm is displayed, such as "Boiler pressure too high".
- **Outgoing**
  The condition for triggering an alarm is no longer fulfilled. The alarm is no longer displayed as the boiler was vented.
- **Acknowledged**
  The operator has acknowledged the alarm.

Alarms without acknowledgment

The following table shows the alarm states for alarms that do not have to be acknowledged:

Icon	Status	Status text	Description
	Active	Incoming	The condition for an alarm is fulfilled. The alarm does not have to be acknowledged.
	Inactive	Normal	The condition for an alarm is no longer fulfilled. The alarm is no longer pending.

Alarms with acknowledgment

The following table shows the alarm states for alarms that have to be acknowledged:

Icon	Status	Status text	Description
	Active	Incoming	The condition for an alarm is fulfilled.
	Inactive, Not acknowledged	Incoming/Outgoing	The condition for an alarm is no longer fulfilled. The operator has not acknowledged the alarm.
	Inactive, Then acknowledged	Incoming/Acknowledged	The condition for an alarm is no longer fulfilled. The operator has acknowledged the alarm after this time.
	Active, acknowledged	Incoming/Outgoing /Acknowledge	The condition for an alarm is fulfilled. The operator has acknowledged the alarm.
	Inactive, Previously acknowledged	Incoming/Acknowledge/Outgoing	The condition for an alarm is no longer fulfilled. The operator acknowledged the alarm while the condition was still fulfilled.

Note

The display text for the states of an alarm is language-specific and configuration-specific.

Locked alarms

You lock an alarm, for example, to prevent an error alarm from impairing the effectivity of your system.

- Locked: The alarm was locked. The alarm assumes its final state without additional state transitions.
- Not locked: The alarm is unlocked. The alarm is visible again in its last state.

Suppressed alarms

You suppress the display of specific alarms, for example, to avoid an excessive burden of information for the plant operator.

- Suppressed manually: The alarm was suppressed manually.
- Suppressed by design: The alarm was suppressed automatically.

See also

Alarm system (Page 707)

Alarm classes (Page 717)

Acknowledging alarms (Page 721)

Acknowledgment model (Page 715)

Alarm components and properties (Page 722)

6.1.4 Acknowledgment model

Overview

The acknowledgment model and the state machine for predefined alarm classes have already been set. You can only set the acknowledgment model and the state machine for user-defined alarm classes. All alarms included in this alarm class are then acknowledged according to this acknowledgment model and the state machine.

The following state machines are available for HMI alarm classes:

- Alarm with single-mode acknowledgment
  This alarm must be acknowledged as soon as the event that triggers the alarm occurs. The alarm remains pending until it is acknowledged.
  The following predefined alarm classes use this state machine: "SystemAlarm", "SystemWarning", "Critical", "Warning", "OperatorInputRequest", "Alarm", "Acknowledgement"
- Alarm with optional single-mode acknowledgment
  This alarm does not require acknowledgement once the event that triggers the alarm has occurred. The alarm is pending until the condition that triggered the alarm is no longer fulfilled.
- Alarm with acknowledgment and confirmation
  The alarm must be acknowledged as soon as the event that triggers the alarm has occurred or the alarm is reset. The alarm also requires a confirmation when the event that triggered the alarm is no longer present. The alarm remains pending until it is acknowledged and confirmed.
  The following predefined alarm classes use this state machine: "AlarmWithReset", "CriticalWithReset", "WarningWithReset".
- Alarm without acknowledgment
  This alarm comes and goes without having to be acknowledged.
  The following predefined alarm classes use this state machine: "SystemNotification", "Notification", "No Acknowledgement".
- Alarm without outgoing status with acknowledgment
  This alarm must be acknowledged as soon as the event that triggers the alarm occurs. The alarm remains pending until it is acknowledged.
  The following predefined alarm classes use this state machine: "SystemAlarmWithoutClearEvent", "SystemWarningWithoutClearEvent".
- Alarm without outgoing status without acknowledgment
  This alarm is displayed until the event that triggered the alarm is no longer pending. The alarm is then no longer displayed in the alarm control.
  The following predefined alarm classes use this state machine: "Information", "Systeminformation", "OperatorInputInformation".
- Alarm without state
  This alarm only has the temporary status "Incoming" and can be seen in the log.

The following state machines are available for HMI alarm classes that are linked to common alarm classes:

- Alarm with single-mode acknowledgment
- Alarm without acknowledgment

Acknowledging and confirming alarms

- Group acknowledgment of alarms in the alarm control
  The alarm control has a "Group acknowledgment" button. This button triggers the acknowledgment of all visible alarms that require acknowledgment and are pending in the alarm control.
- Single acknowledgment of alarms in the alarm control
  The alarm control has a "Single acknowledgment" button. This button triggers the acknowledgment of individual alarms selected in the alarm control.
- Single confirmation of alarms with acknowledgment and confirmation in the alarm control.
  The alarm control has a "Single confirm" button. The alarm with the state machine "Alarm with acknowledgment and confirmation" is individually confirmed with this button after it has been acknowledged with group acknowledgment or single acknowledgment beforehand and is outgoing.

Note

If the "Show recent" button is pressed, the most recent alarm is always shown first. Group acknowledgment is only executed for the visible alarms.

See also

Alarm system (Page 707)

Configuring alarm acknowledgment (Page 749)

Alarm components and properties (Page 722)

Acknowledging alarms (Page 721)

Alarm states (Page 714)

Alarm classes (Page 717)

Alarm control (Page 309)

6.1.5 Alarm classes

Introduction

Many alarms occur in a plant. These are all of different importance. You can assign the alarms of your project to alarm classes to clearly show the operator which of the alarms are most important.

Description

Every alarm must be assigned to an alarm class when you create new alarms. The alarm class hereby defines the appearance and the acknowledgment model of the alarm (single-mode acknowledgment, acknowledgment and confirmation, no acknowledgment).

A new alarm class with mandatory acknowledgment is generated in WinCC. Predefined alarm classes are available for each device.

Examples of how to use alarm classes

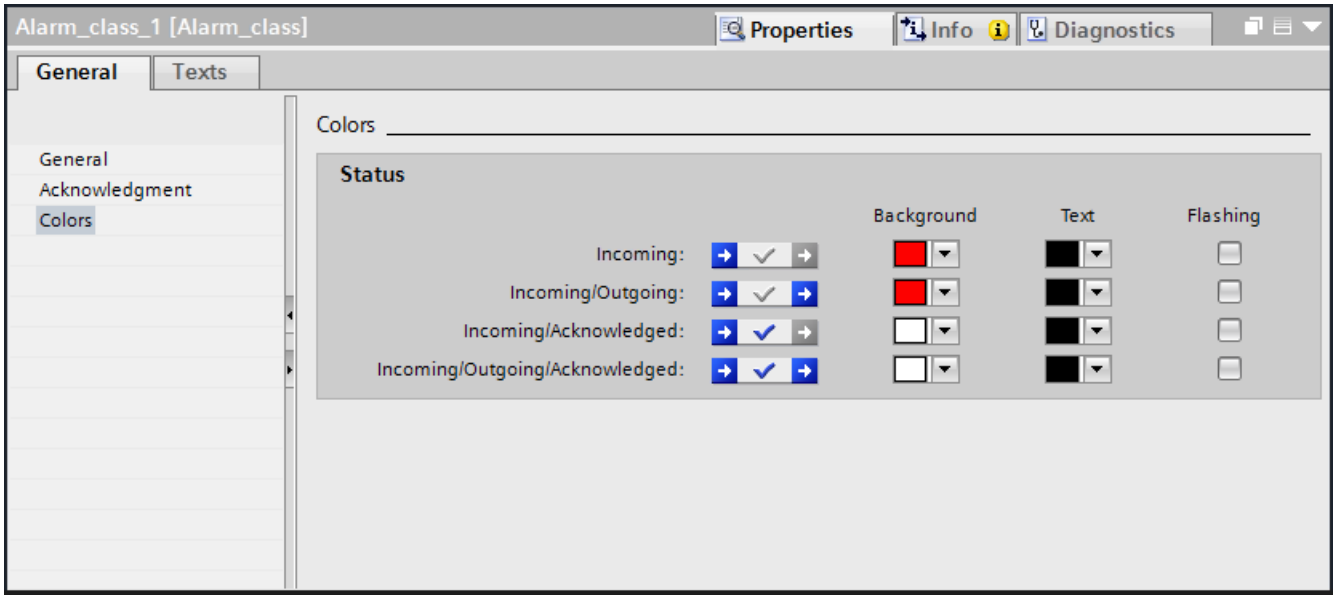
- The alarm class of the alarm "Fan 1 speed in upper tolerance range" is "Warning". The alarm is displayed with a yellow background. The alarm requires acknowledgment.
- The alarm "Speed of fan 2 has exceeded upper warning range" is assigned to the "Alarm" alarm class. The alarm is displayed with a red background and flashes at high frequency in runtime. The alarm is displayed until the alarm is gone and the operator has acknowledged it.

Using alarm classes

Use the following alarm classes to define the state machines and appearance of alarms for your project:

- Predefined alarm classes
  You cannot delete predefined alarm classes and edit them only to a limited extent. Predefined alarm classes are available under "HMI alarms > Alarm classes".
- Custom alarm classes
  You can create new alarm classes under "HMI alarms > Alarm classes", configure how you want the alarms to be displayed, and define an acknowledgment model for alarms of this alarm class. The possible number of custom alarm classes depends on which runtime is used in your project.
- Common alarm classes
  Common alarm classes are displayed under "Common data > Alarm classes" in the project tree and can be used for the alarms of an HMI device. Common alarm classes are used in STEP 7 for controller alarms. If required, create additional common alarm classes in WinCC. Common alarm classes are divided into predefined and user-defined common alarm classes. The predefined common alarm classes are "Acknowledgement" (for alarms with acknowledgment) and "No Acknowledgement" (for alarms without acknowledgment).

For each alarm class (including predefined alarm classes), you can configure the font color, background color and flashing for the alarm states "Incoming", "Incoming/outgoing", "Incoming/acknowledged", "Incoming/outgoing/acknowledged":



Predefined alarm classes

The following predefined alarm classes are available under "Alarm classes" in the "HMI alarms" editor.

System events:

- "SystemInformation"
  Alarms of this alarm class have no "Outgoing" state and do not require acknowledgment. This alarm class is only used for logging.
- "SystemNotification"
  Alarms in this alarm class do not require acknowledgment.
- "SystemWarning"
  Alarms in this alarm class must be acknowledged.
- "SystemWarningWithoutClearEvent"
  Alarms of this alarm class have no "Outgoing" state and must be acknowledged.
- "SystemAlarm"
  Alarms in this class must be acknowledged.
- "SystemAlarmWithoutClearEvent"
  Alarms of this alarm class have no "Outgoing" state and must be acknowledged.

Standard alarms:

- "Acknowledgement"
  Alarms in this class must be acknowledged. The alarm class "Acknowledgement" is linked to the predefined common alarm class "Acknowledgement".
- "No Acknowledgement"
  Alarms in this alarm class do not require acknowledgment. The alarm class "No Acknowledgement" is linked to the predefined common alarm class "No Acknowledgement".

- "Information"
  Alarms of this alarm class have no "Outgoing" state and do not require acknowledgment. This alarm class is only used for logging.
- "OperatorInputInformation"
  Alarms of this alarm class have no "Outgoing" state and do not require acknowledgment. The alarm class "OperatorInputInformation" is designed to show the reports that are relevant for an audit. This alarm class is only used for logging.
- "Notification"
  Alarms in this alarm class do not require acknowledgment. The alarm class "Notification" alarm class is designed to show irregular states and routines in the process.
- "OperatorInputRequest"
  Alarms in this alarm class must be acknowledged.
- "Warning"
  Alarms in this alarm class must be acknowledged.
- "WarningWithReset"
  Alarms in this alarm class must be acknowledged. This alarm class must also be reset. The locked alarm is unlocked during the reset.
- "Alarm"
  Alarms in this class must be acknowledged. The alarm class "Alarm" is designed to show critical or dangerous states or limit violations in the process.
- "AlarmWithReset"
  Alarms in this alarm class must be acknowledged. This alarm class must also be reset. The locked alarm is unlocked during the reset.
- "Critical"
  Alarms in this alarm class must be acknowledged. The alarm class "Critical" is designed to show critical faults in the plant, for example, "Motor temperature too high". This alarm and its alarm states are included in the log.
- "CriticalWithReset"
  Alarms in this alarm class must be acknowledged. This alarm class must also be reset. The locked alarm is unlocked during the reset. The alarm class "CriticalWithReset" is designed to show critical faults in the plant, for example, "Motor temperature too high".

Note

The alarm classes whose names contain "System" are designed to show the states of the device and the controllers, for example, to provide information on operating errors or faults in communication.

Note

The predefined alarm classes are write-protected and cannot be deleted. You can, however, change the preset background and foreground colors and font colors if necessary. If required, you can also change the name of the predefined alarm classes "Acknowledgement" and "No Acknowledgement". The name of the linked predefined common alarm classes "Acknowledgement" and "No Acknowledgement" is not changed. You cannot change the name of the linked predefined common alarm classes, not even under "Common data > Alarm classes". Common alarm classes are automatically added for the non-intergrated mode.

Custom alarm classes

The properties of this alarm class are defined in the configuration. You assign a name, a state machine, a priority and, if necessary, a log to the alarm class. You also define the display of the user-defined alarm using text and background colors.

See also

Creating alarm classes (Page 726)

Using common alarm classes (Page 729)

Alarm system (Page 707)

Acknowledging alarms (Page 721)

Alarm states (Page 714)

Acknowledgment model (Page 715)

Alarm components and properties (Page 722)

6.1.6 Acknowledging alarms

Introduction

To make sure that an alarm was noticed by the plant operator, configure this alarm so that it is displayed until acknowledged by the operator. Alarms that indicate critical or hazardous states in the process have to be acknowledged.

The acknowledgment of an alarm is an event that is logged and reported. Acknowledging an alarm in the "Incoming" state changes the alarm state from "Incoming" to "Acknowledged". When the operator acknowledges an alarm, they confirm that they have processed the state that triggered the alarm.

Acknowledging an alarm

The operator acknowledges in runtime an alarm via the alarm control buttons.

See also

Alarm system (Page 707)

Configuring alarm acknowledgment (Page 749)

Acknowledgment model (Page 715)

Alarm classes (Page 717)

Alarm states (Page 714)

Alarm components and properties (Page 722)

6.1.7 Alarm components and properties

Overview

The following table shows the basic components of alarms that you can configure in WinCC:

Alarm class	Alarm number	Time	Date	State machine	Alarm text	Info text	Trigger tag	Limit value
Warning	1	11:09:14	06.08.2017	Alarm with single-mode acknowledgment	Maximum speed reached	This alarm is ...	speed_1	27

Alarm class

The alarm class of an alarm determines whether the alarm has to be acknowledged.

The alarm class defines the following for an alarm:

- State machine/acknowledgment model
- Appearance in runtime (e.g. color)
- Log in which an alarm is logged
- Priority

Alarm number

An alarm is identified by an alarm number (ID). The alarm number is assigned by the system for internally managing an alarm. You can change the alarm number to a sequential alarm number, if necessary, to identify alarms associated in your project.

An alarm number must only be used once on a device.

Note

Discrete alarms and analog alarms can receive an identical alarm number from the system. The alarm number can be customized on request.

Note

When adapting alarm numbers, observe the inter-project uniqueness of the alarm number.

The system event number overrides a custom alarm number. If using the system event number for a custom alarm, change the custom alarm number accordingly.

Time and date

Every alarm has a time stamp that shows the time and date at which the alarm was triggered.

State machine

An alarm has the state machine or the acknowledgment model of the alarm class.

The state machine is how an alarm is displayed in various states and processed from by the system.

Alarm states

An alarm always has a specific alarm state in runtime. The operator analyzes the process execution based on the alarm states.

Alarm text

The alarm text describes the cause of the alarm.

The alarm text can contain output fields for current values. The values you can insert depend on the runtime in use. The value is retained at the time at which the alarm status changes.

Info text

You can configure a separate info text for each alarm; the operator can display this info text in runtime.

Trigger tag

A tag is assigned to each alarm as trigger. The alarm is raised when this trigger tag meets the defined condition, e.g. when its state changes or it exceeds a limit.

Limit value

Analog alarms indicate limit violations. Depending on the configuration, WinCC outputs the analog alarm as soon as the trigger tag exceeds or undershoots the limit value.

Computer

Operator input alarms have the "Computer" column in the alarm lists. The computer name is displayed for local alarms and the IP address for alarms from the web client.

Users

The user acknowledges the alarm. If an empty user name is transferred to an alarm, the alarm displays no user name.

Duration

The time interval between the triggering of the alarm and its last state change in nanoseconds.

See also

Alarm system (Page 707)

Acknowledgment model (Page 715)

Alarm states (Page 714)

Alarm classes (Page 717)

Acknowledging alarms (Page 721)

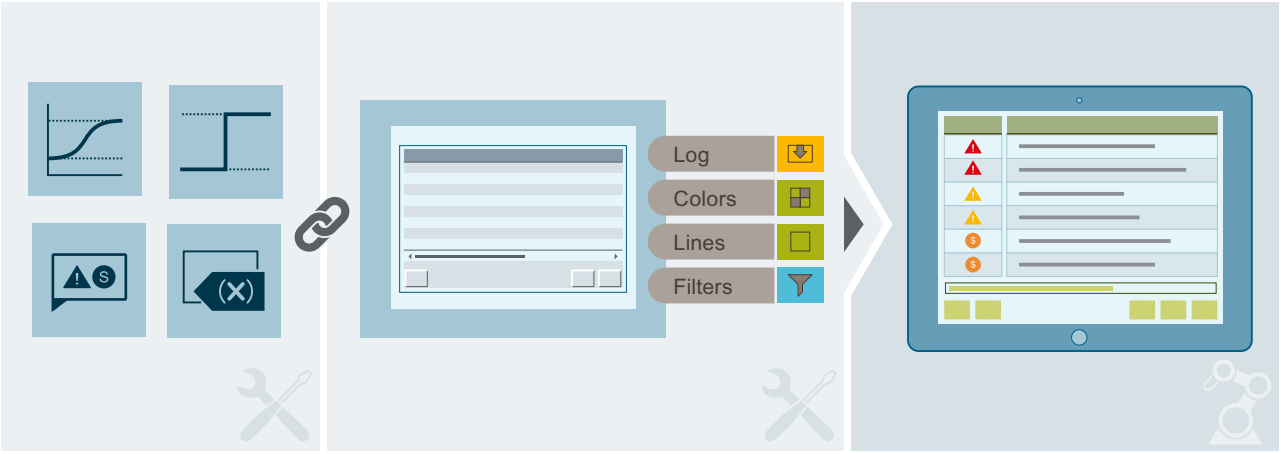
6.2 Configuring alarms

6.2.1 Workflow for configuring alarms

Steps to configure alarms

You configure alarms in the following stages:

1. Configuring alarm classes
   In the "HMI alarms" editor, adapt the predefined alarm classes or configure your own alarm classes. You use the alarm class to define the state machine of an alarm and how the alarm is displayed in runtime.
2. Creating trigger tags
   Create trigger tags in the "HMI tags" editor and configure the trigger bit for discrete alarms or the range limits for analog alarms.
3. Creating alarms
   Create user-defined alarms in the "HMI alarms" editor and assign the alarm classes and the tags to be monitored.
   Under "Runtime settings > Alarms", adapt the status texts or activate the use of controller alarms.
4. Configuring the display of alarms
   Configure an alarm control in the "Screens" editor.
5. Logging alarms
   In the "Logs" editor, create an alarm log and assign the log to an alarm class in the "HMI alarms" editor.



Additional configuration tasks

Additional tasks could be necessary for configuring alarms, depending on the requirements of your project:

- Adapting status texts
  You specify the status texts under "Runtime settings > Alarms".
- Editing system events
  You edit system events in the "HMI alarms" editor or under "Languages & Resources > Project texts". In the "Category" column you can recognize a system event by the name "HMI system event".
- Activating controller alarms
  For integrated operation of a project in STEP 7, specify the controller alarms to be displayed on your HMI device in the alarm settings.

Note

WinCC only supports controller alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

See also

Creating alarm classes (Page 726)

Using common alarm classes (Page 729)

Configuring discrete alarms (Page 734)

Configuring analog alarms (Page 738)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

Configuring multilingual alarm texts (Page 747)

Editing system events (Page 748)

Configuring alarm acknowledgment (Page 749)

Filtering controller alarms via display classes (Page 748)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Configuring automatic update of controller alarms on the HMI device (Page 797)

Configuring the display of system diagnostic alarms (Page 767)

6.2.2 Creating alarm classes

Introduction

Create alarm classes to define the type of acknowledgment, the display of the alarm in runtime for an alarm. You assign the individual alarms to the alarm classes.

Create alarm classes in the "Alarm classes" tab of the "HMI alarms" editor. Some default alarm classes are already created for every project. You can create additional custom alarm classes. The alarm class also specifies if and how the operator has to acknowledge alarms of this alarm class.

System alarm classes are write-protected and cannot be deleted. You can, however, change the preset background and foreground colors and font colors if necessary. All system alarm classes have the word "System" in their name and are to be used for system-defined alarms.

Note

Alarm colors are editable depending on the state machine. Some colors are not editable for certain state machines.

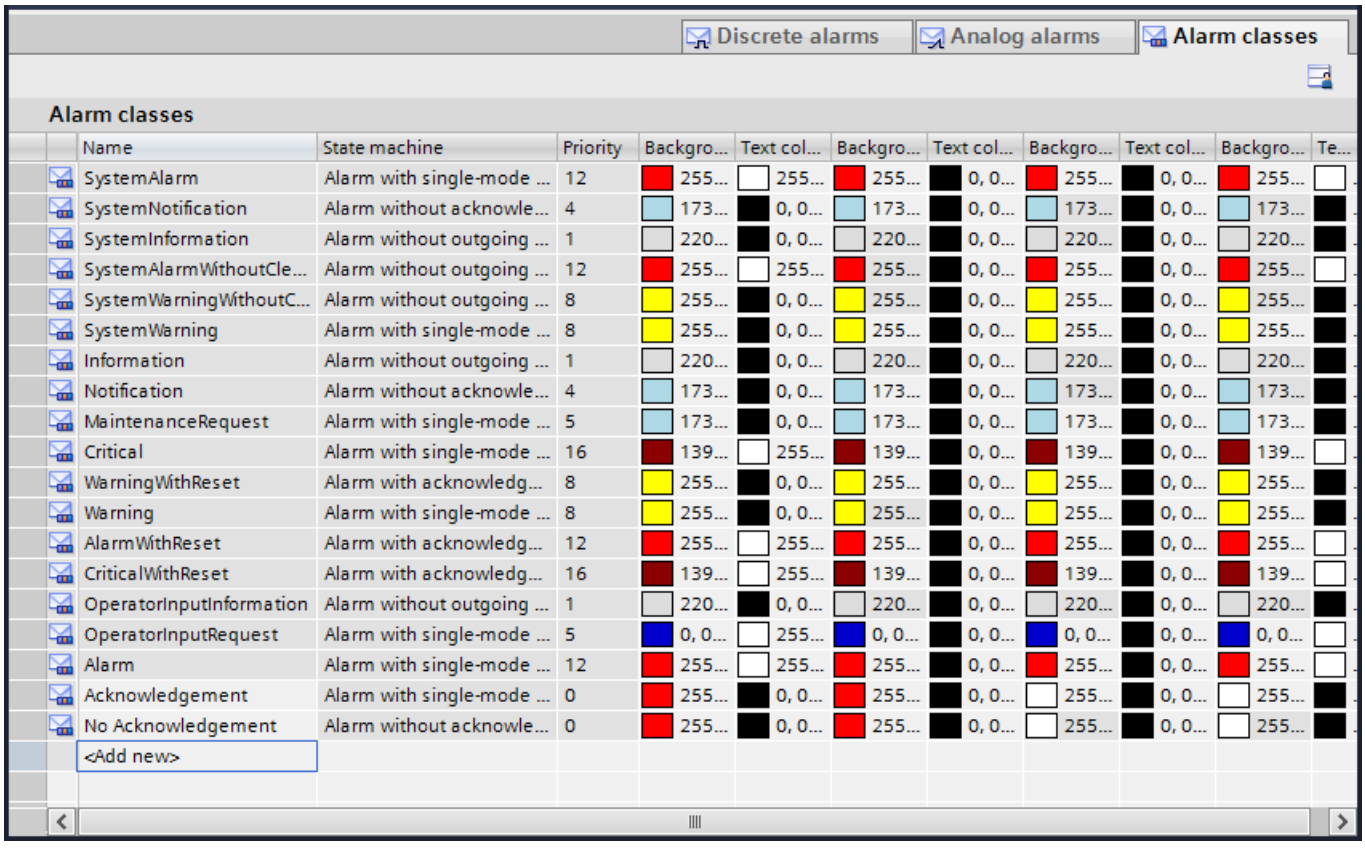
Requirement

- The "HMI alarms" editor is open.
- The Inspector window is open.

Procedure

To create an alarm class, proceed as follows:

1. Click the "Alarm classes" tab.
   A table of the pre-defined alarm classes is shown below:



2. Double-click "<Add>" in the table.
   A new alarm class is created. Each new alarm class is automatically assigned a static ID.
   The properties of the new alarm class are shown in the Inspector window.
3. Configure the alarm class under "Properties > General" in the Inspector window:
   - Enter a name for the alarm class.
   - Set the priority of the alarm class.
4. In the Inspector window, set the state machine of the alarm class.
   See Configuring alarm acknowledgment (Page 749).
5. You can also change the default background color as well as the text color and the settings for flashing under "Properties > Colors" in the Inspector window.
   These settings define how alarms from this alarm class are displayed in runtime.

Note

For alarm colors to be displayed in runtime, the "Use alarm colors" option must be enabled in the properties of the alarm control in the Inspector window. This option is enabled by default.

"Acknowledgment" alarm class

A alarm of the "Acknowledgment" alarm class passes through the following alarm states:

- Incoming
  When the alarm occurs.
- Incoming/Outgoing
  When the alarm is outgoing.
- Normal
  After resetting the value on the PLC and through acknowledgment of the operator.

"No Acknowledgement" alarm class

An alarm of the "No acknowledgment" alarm class passes through the following alarm states:

- Incoming
  When the alarm occurs.
- Normal
  When the alarm is outgoing.

"AlarmWithReset" alarm class

An alarm of the "No acknowledgment" alarm class passes through the following alarm states:

- Incoming
  When the alarm occurs.
- Incoming/outgoing or incoming/acknowledged
  When the alarm is outgoing.
- Incoming/acknowledged/outgoing or incoming/outgoing/acknowledged
  After resetting of the alarm by the operator.
- Normal
  After performing all actions.

See also

Alarm classes (Page 717)

Using common alarm classes (Page 729)

Workflow for configuring alarms (Page 724)

Configuring discrete alarms (Page 734)

Configuring analog alarms (Page 738)

6.2.3 Using common alarm classes

Introduction

Common alarm classes are displayed under "Common data > Alarm classes" in the project tree and can be used for the alarms of an HMI device. Common alarm classes are used in STEP 7 for controller alarms. If required, create additional common alarm classes in WinCC. Common alarm classes are divided into predefined and user-defined common alarm classes. The predefined common alarm classes are "Acknowledgement" (for alarms with acknowledgment) and "No Acknowledgement" (for alarms without acknowledgment).

When you create an HMI device, the system creates an alarm class for each existing common alarm class under "HMI alarms > Alarm classes" which is linked to the common alarm class. If you have created an HMI device and create a common alarm class, the system creates an alarm class for the created common alarm class under "HMI alarms > Alarm classes" which is linked to the common alarm class. If you change all properties for a created common alarm class, the system changes the properties "State machine" and "Priority" of the linked alarm class according to your changes to the "Acknowledgment" and "Priority" properties of the common alarm class. However, your changes to the "Name" and "Display name" properties of the common alarm class have no effect on the properties of the linked alarm class. When you delete a common alarm class, the linked alarm class is also deleted.

Note

By the "Common alarm class" property of an alarm class, you can see whether the alarm class is linked with a common alarm class and, if so, with which common alarm class. To see the property in the "HMI alarms" editor in the "Alarm classes" tab, enable there the "Common alarm class" column using the shortcut menu of the column titles. To see the property in the Inspector window, select an alarm class under "HMI alarms > Alarm classes" and click "General" in the Inspector window.

State machine

Common alarm classes have one of the following state machines:

- Alarm with single-mode acknowledgment
- Alarm without acknowledgment

You configure the state machine directly in the "Alarm classes" editor. Configuration at the linked HMI alarm class in the "HMI alarms" editor is not possible.

Note

For the state machine "Alarm without acknowledgement", the background color configured for the state is ignored in the "Arrived/Gone" state.

Requirements

- You have created a project.

Creating common alarm class

To create a common alarm class, proceed as follows:

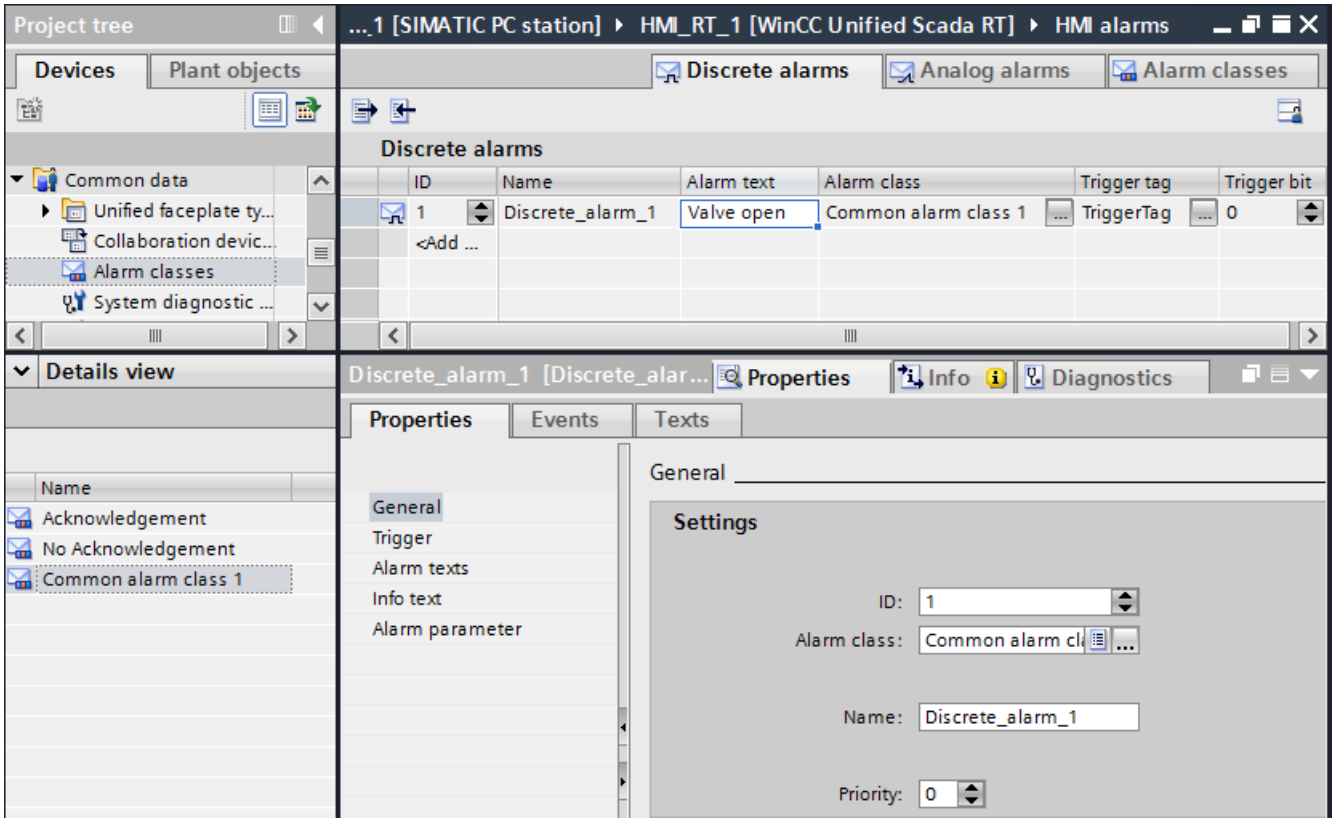
1. Double-click "Common data > Alarm classes" in the project tree.
   The "Alarm classes" editor opens in the working area.
2. To create a common alarm class, double-click in the first empty line of the table editor.
3. Specify the name, display name and priority of the common alarm class and enable the mandatory acknowledgment, if required.
   A common alarm class is created. The system also creates an alarm class under "HMI alarms > Alarm classes", which is linked to the common alarm class. The linked alarm class gets from the system the same name and priority as the common alarm class. If you have enabled the mandatory acknowledgment for the common alarm class, the linked alarm class gets the state machine "Alarm with single-mode acknowledgment" from the system, otherwise the state machine "Alarm without acknowledgment". The defined display name of the common alarm class has no effect on the properties of the linked alarm class.
4. If required, change the name of the alarm class that is linked to the common alarm class under "HMI alarms > Alarm classes".
   If you change the name of the linked alarm class, the common alarm class name is not changed by the system.

Assign alarms to a common alarm class

Proceed as follows to assign an analog or discrete alarm to a common alarm class:

1. In the "HMI alarms" editor, select the alarm that you want to assign to the common alarm class.
2. Click "General" in the Inspector window.
3. Click "Common data > Alarm classes" in the project tree. Alternatively, click "HMI Alarms" in the project tree.
   In the first case, the common alarm class is selected in the detail view. In the second case, the detail view shows the alarm class, which is linked to the common alarm class.

4. Select the common alarm class or alternatively the linked alarm class in the detail view.
5. Drag the common alarm class or alternatively the linked alarm class to the "Alarm class" field or "Alarm class" column in the working area of the Inspector window of the alarm.
   In both cases, the alarm is assigned to the alarm class which is linked to the common alarm class.



Changing a common alarm class

To change a common alarm class, proceed as follows:

1. Double-click "Common data > Alarm classes" in the project tree.
   The "Alarm classes" editor opens in the working area.
2. If necessary, change the name of the created common alarm class.
   The changed name of the common alarm class has no effect on the name of the alarm class which is linked to the common alarm class.
3. If necessary, change the display name of the common alarm class.
   The changed display name of the common alarm class has no effect on the properties of the linked alarm class.

4. If required, enable or disable the mandatory acknowledgment of the common alarm class. If you enable the mandatory acknowledgment, the system changes the state machine of the linked alarm class to "Alarm with single-mode acknowledgment". If you disable the mandatory acknowledgment, the system changes the state machine of the linked alarm class to "Alarm without acknowledgment".
5. If necessary, change the priority of the common alarm class.
   The system changes the priority of the linked alarm class according to your change to the priority of the common alarm class.

Note

You can only change the display names for a predefined common alarm class. The changed display name of a predefined common alarm class has no effect on the properties of the alarm class which is linked to the predefined common alarm class.

Deleting a common alarm class

To delete a common alarm class, proceed as follows:

1. Double-click "Common data > Alarm classes" in the project tree.
   The "Alarm classes" editor opens in the working area.
2. Select the created common alarm class that you want to delete.
3. Select the "Delete" entry from the shortcut menu.
   The system deletes the common alarm class and the alarm class linked with the common alarm class.
4. If an analog or discrete alarm has been assigned to the deleted linked alarm class, assign another alarm class to the alarm. Otherwise a compile error will be generated.

Note

You cannot delete predefined common alarm classes and the alarm classes that are linked with them.

See also

Alarm classes (Page 717)

Creating alarm classes (Page 726)

Workflow for configuring alarms (Page 724)

Configuring discrete alarms (Page 734)

Configuring analog alarms (Page 738)

6.2.4 Configuring state texts of alarms

Introduction

The texts for the states of alarms in runtime are displayed in the alarm view in the "Status Text" column. You specify the state texts of alarms in the runtime settings.

Note

If no alarm state texts are defined, an error is generated during compiling.

Requirement

- The alarm view has been configured.

Procedure

To configure the state texts of alarms, follow these steps:

1. Open the "Runtime settings" of the HMI device.
2. Specify the state texts of alarms in runtime under "Alarms > State texts":

Field	Description
Incoming	Text for incoming alarms when changing to the operating state to be reported The condition for an alarm is fulfilled.
Incoming/outgoing	Text for an outgoing alarm The condition for an alarm is no longer fulfilled.
Incoming/acknowledged	Text for an acknowledged and outgoing alarm The condition for an alarm is no longer fulfilled.
Incoming/acknowledged/outgoing	The condition for an alarm is no longer fulfilled. Text for an incoming, acknowledged and outgoing alarm.
Incoming/outgoing/acknowledged	The condition for an alarm is no longer fulfilled. Text for an incoming, outgoing and acknowledged alarm
Removed	Text for alarms in "Removed" state. Only controller alarms can have this state. The state text is only displayed in the alarm log. If the HMI connection between HMI device and controller is disconnected and then re-established, the status text is displayed.
Normal	The condition for an alarm is no longer fulfilled. The alarm is no longer pending. The alarm is acknowledged if required by the alarm class. The alarm is reset by single confirm if required by the alarm class.

3. Change the status texts as required. Keep in mind that these texts are not language specific. In Runtime, always the text configured in engineering is always displayed, independent of the current Runtime language.

4. Select the alarm view in the "Screens" editor.
5. To display the column "Status text" in the Alarm view, select the property "Visibility" in the Inspector window under "Properties > Alarm view > Columns > [...] Status text alarm column".

See also

Configuring an alarm control (Page 753)

6.2.5 Configuring discrete alarms

6.2.5.1 Configuring discrete alarms

Introduction

Discrete alarms triggered by the PLC indicate status changes in a plant. A discrete alarm is triggered by a specific value (bit) of a tag.

Imagine, for example, that the state of a valve is to be monitored during operation. The two possible valve states are "opened" and "closed". In this case, a discrete alarm is configured for each valve state. A discrete alarm containing the following alarm text, for example, is output when the state of this valve changes: "Valve closed".

Note

By default, each new discrete alarm is assigned the alarm class "Alarm". You can then alter the alarm class as required.

Requirement

- The "HMI alarms" editor is open.
- The Inspector window is open.

Procedure

To configure a discrete alarm, proceed as follows:

1. Click the "Discrete alarms" tab.
2. To create a new discrete alarm, double-click on "<Add>" in the table.
   A new discrete alarm is created.

3. Assign a name for the discrete alarm.

Note

The name of a discrete alarm can contain up to 128 characters.

4. To configure the alarm, select "Properties > General" in the Inspector window:
   - Edit the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").

Note

You can use the priority to sort or filter the alarms in the alarm control. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display area in a single-line alarm control.

If you filter the alarm control by priority "16", only the alarms with priority "16" will appear.

For alarms with priority "0", the priority of the alarm class applies.

The priority of the alarm when displayed in runtime takes precedence over the priority of the alarm class.

Tips for an efficient procedure

- Large machines and plants have a large number of alarm sources that can trigger various types of alarms. It makes sense to structure the alarm system so that the user can keep track of this wide range. One suitable method available here is alarm prioritization. The criteria for assigning the priority value and/or the alarm class are importance and urgency. The priority of the alarm can also be based on the potential impact (system downtime, loss of production, production delay, etc.). If multiple alarms are output, the system can suggest the order in which they should be handled on the basis of priorities.
- You create discrete alarms together with the trigger tags and edit them in the "HMI tags" editor. You create tags in the usual way. Then click <Add> in the table on the "Discrete alarms" tab at the bottom of the work area. A new discrete alarm is created for the tag. If you have selected the wrong data type, the tag will be highlighted in the discrete alarm. If you delete, move or copy objects in the "HMI tags" editor, these changes also take effect in the "HMI alarms" editor. The configured discrete alarms are created in the "HMI tags" editor and displayed in the "HMI alarms" and "HMI tags" editors.
  If you are using an array as a data type for managing alarms, you can assign a discrete alarm to the individual array elements. The alarms assigned to the array elements are displayed under "Discrete alarms" or when you select the array. The alarms have the position within the array as an extension, e.g. "HMI_Tag_1[2]".
- Supplementary information about individual alarms ensures that faults are localized and cleared quickly.

See also

Discrete alarms (Page 710)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

Workflow for configuring alarms (Page 724)

Configuring alarm acknowledgment (Page 749)

Creating alarm classes (Page 726)

Using common alarm classes (Page 729)

Configuring multilingual alarm texts (Page 747)

Editing system events (Page 748)

Configuring analog alarms (Page 738)

Creating internal tags (Page 629)

6.2.5.2 Configure trigger

You can define trigger properties for discrete alarms. You can specify a trigger tag, a trigger bit and a trigger mode for an alarm.

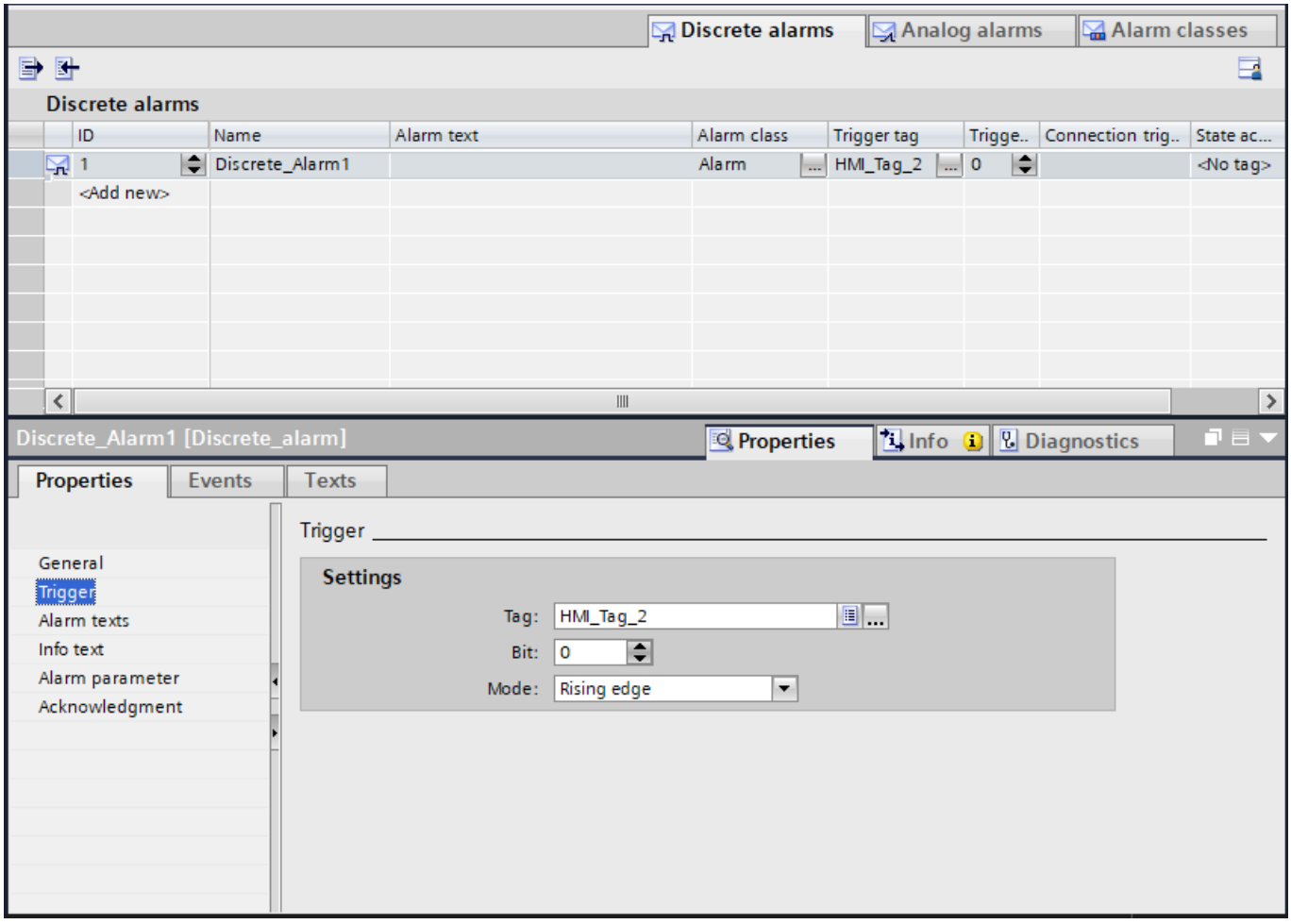
Requirement

- The "HMI alarms" editor is open.
- A discrete alarm has been created.

Procedure

To define trigger properties, follow these steps:

1. Select the discrete alarm.
2. Click on "Trigger" under "Properties" in the Inspector window.
   The "Trigger" window opens.
3. Select the tag and the bit that are to trigger the alarm. You can also specify whether to trigger the alarm at a rising or falling edge.



Note

Restrictions for trigger bits

Each use of a trigger tag requires a different trigger bit. The availability of trigger bits depends on the data type of the trigger tag.

Result

Trigger tag, trigger bit and trigger mode have been specified for the selected discrete alarm.

6.2.5.3 Sending alarm acknowledgments to the PLC

Requirement

- The "HMI alarms" editor is open.
- The required alarm has been created and assigned to an alarm class requiring acknowledgment.

Core statement

To configure that acknowledgment of an alarm is sent to the PLC, follow these steps:

1. In the "HMI alarms" editor, click the "Discrete alarm" tab and select the discrete alarm.
2. In the Inspector window, select "Properties > Properties > Acknowledgment".
3. Under "Status tag", select the tag and the bit set by the alarm acknowledgment function.
4. Under "Control tag", select the tag and the bit set by the alarm acknowledgment function.

Note

The HMI device and PLC only have read access to the acknowledgment tag memory area.

Result

If the operator acknowledges the alarm in Runtime, the operating step is forwarded to the PLC.

6.2.6 Configuring analog alarms

6.2.6.1 Configuring analog alarms

Introduction

You configure analog alarms to display limit violations. You have defined in advance a limit value for the trigger tag and the trigger mode. An analog alarm is triggered depending on which mode you have defined, for example, when the value is higher than, lower than or the same as the defined value.

If the speed of a motor drops below a certain value, for example, an analog alarm is triggered. This alarm could contain the following text: "Motor speed is too low".

Note

By default, each new analog alarm is assigned the alarm class "Alarm". You can then alter the alarm class as required.

Requirement

- The "HMI alarms" editor is open.
- The Inspector window is open.

Procedure

To configure an analog alarm, proceed as follows:

1. Click the "Analog Alarms" tab.
2. To create a new analog alarm, double-click in the table on "<Add>".
   A new analog alarm is created.
3. To configure the alarm, select "Properties > General" in the Inspector window:
   - Edit the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").

Note

You can use the priority to sort or filter the alarms in the alarm control. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display area in a single-line alarm control.

If you filter the alarm control by priority "16", only the alarms with priority "16" will appear.

For alarms with priority "0", the priority of the alarm class applies.

The priority of the alarm when displayed in runtime takes precedence over the priority of the alarm class.

Tips for an efficient procedure

- Large machines and plants have a large number of alarm sources that can trigger various types of alarms. It makes sense to structure the alarm system so that the operator can keep track of this wide range. One suitable method available here is alarm prioritization. The criteria for assigning the priority value and/or the alarm class are importance and urgency. The priority of the alarm can also be based on the potential impact (system downtime, loss of production, production delay, etc.). If multiple alarms are output, the system can suggest the order in which they should be handled on the basis of priorities.
- You create analog alarms together with the trigger tags and edit them in the "HMI tags" editor. You create tags in the usual way and configure the range values of the tags. Then click <Add> in the table on the "Analog alarms" tab at the bottom of the work area. A new analog alarm is created for the tag. If you have selected the wrong data type, the tag will be highlighted in the analog alarm. If you delete, move or copy objects in the "HMI tags" editor, these changes also take effect in the "HMI alarms" editor. The configured analog alarms are created in the "HMI tags" editor and displayed in the "HMI alarms" and "HMI tags" editors.
- Supplementary information about individual alarms ensures that faults are localized and cleared quickly.

See also

Analog Alarms (Page 709)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

Workflow for configuring alarms (Page 724)

Configuring alarm acknowledgment (Page 749)

Creating alarm classes (Page 726)

Using common alarm classes (Page 729)

Configuring discrete alarms (Page 734)

Configuring multilingual alarm texts (Page 747)

Creating internal tags (Page 629)

6.2.6.2 Configure trigger

Trigger properties can be defined for analog alarms. You can specify a trigger tag, a trigger bit and a trigger mode for an alarm.

Requirement

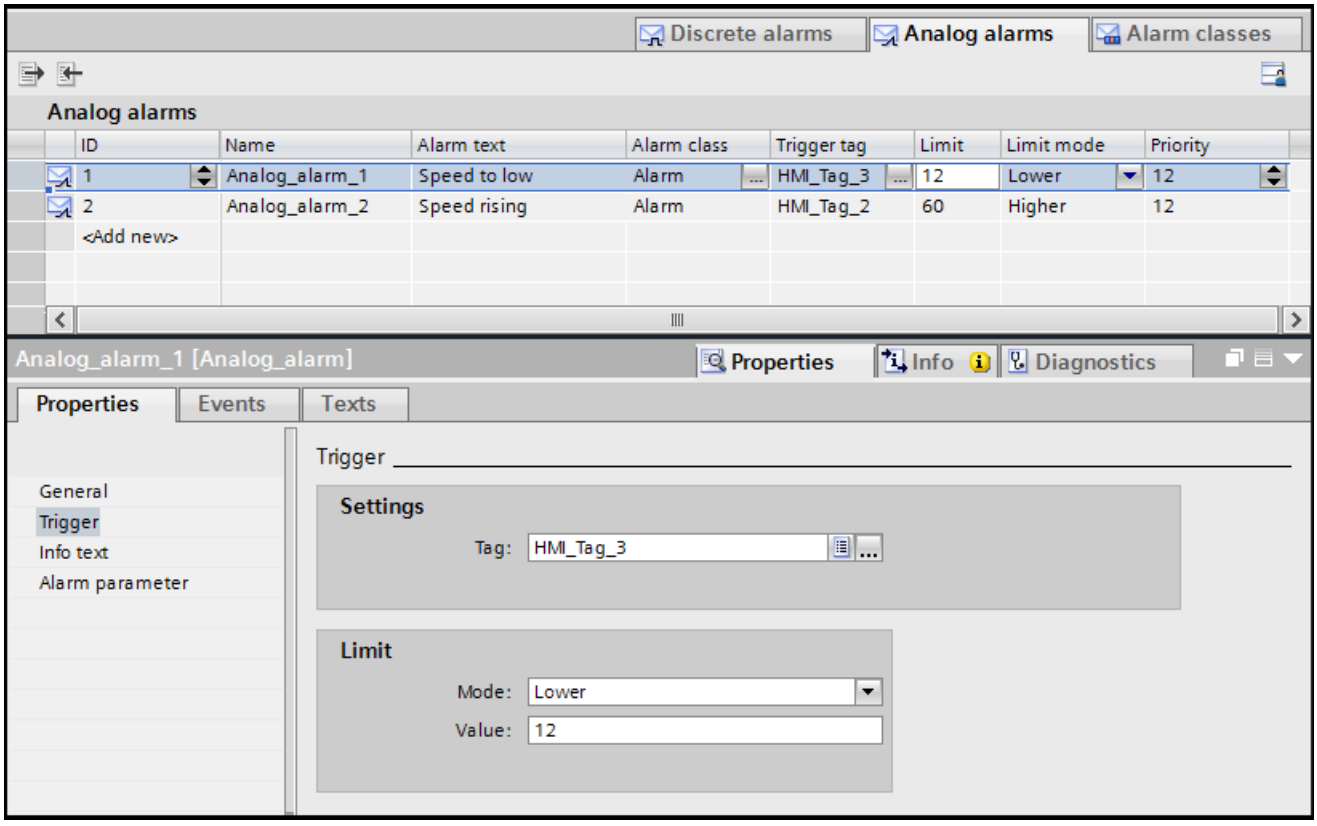
- The "HMI alarms" editor is open.
- A discrete alarm has been created.

Requirement

To define trigger properties, follow these steps:

1. Select the discrete alarm.
2. Click on "Trigger" under "Properties" in the Inspector window.
   The "Trigger" window opens.
3. Specify the trigger tags and limits. Use one of the following data types: "Int", "Real", "LReal", "SInt", "USInt", "UInt", "UDInt" and "ULInt".

4. Select the trigger mode in the "Mode" field:
   - "Less": The alarm is triggered if the limit is undershot.
   - "Greater": The alarm is triggered if the limit is exceeded.
   - "Equal": The alarm is triggered when the limit is reached.
   - "Not equal": The alarm is triggered if the limit is not reached.
   - "Less or equal": The alarm is triggered if the limit is undershot or reached.
   - "Greater or equal": The alarm is triggered if the limit is exceeded or reached.
5. You can create additional limits for the alarm, if necessary. Note the following information:
   - A tag is monitored using only one alarm type. You should therefore create either analog alarms **or** discrete alarms for a tag.
   - If the object included in the selection does not yet exist, create it in the object list and change its properties later.



Note

Do not use trigger tags for anything else.

Result

Trigger tags and limits have been defined for the selected analog alarm.

6.2.7 Integrating OPC UA server alarm instances

You have the option of integrating alarm instances from an OPC UA server into your Runtime project. Proceed as described in the section Integrating OPC UA server alarm instances into a Unified client (Page 7046).

6.2.8 Configuring alarm texts

Introduction

For an alarm, you can configure up to ten alarm texts: one alarm text and up to nine additional texts. If required, you can insert output fields for displaying alarm parameters in each alarm text. Each alarm text contains up to 512 characters.

Requirement

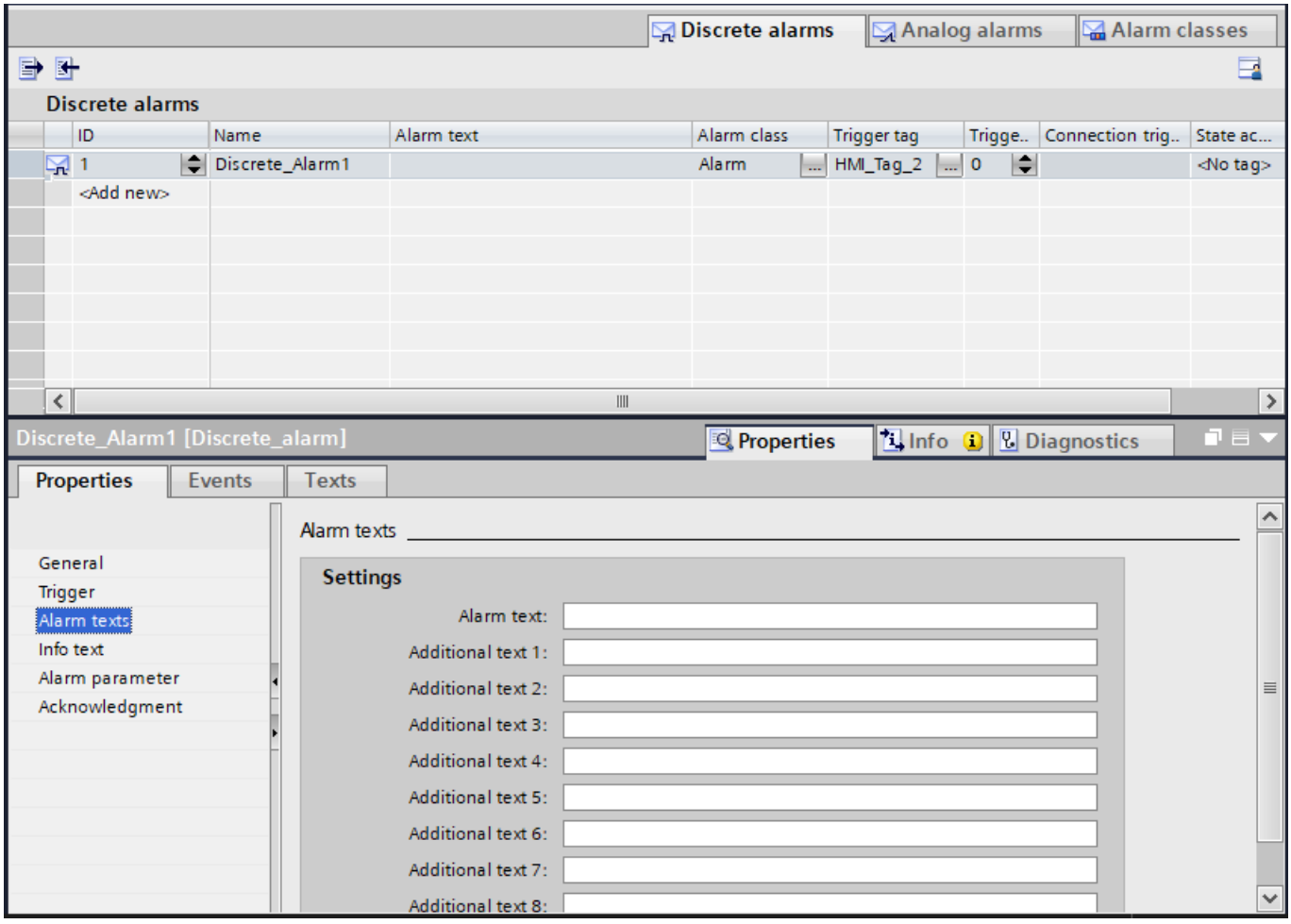
- An alarm has been created.
- The alarm control has been configured.

Procedure

To configure alarm texts, follow these steps:

1. Select the discrete or analog alarm in the "HMI alarms" editor.
2. Enter an alarm text under "Properties > Properties > Alarm texts > Settings > Alarm text" in the Inspector window.
3. If necessary, insert parameter output fields in the alarm text via the "Insert parameter field" shortcut menu command.
4. Enter additional alarm texts in the fields for additional texts under "Properties > Properties > Alarm texts" in the Inspector window.
5. If necessary, insert parameter output fields in the other alarm texts using the "Insert parameter field" shortcut menu command.

6. Select the alarm control in the "Screens" editor.
7. To display the alarm texts in runtime, enable the required columns of the columns numbered 10 to 19 in the Inspector window under "Properties > Properties > Alarm control > Columns".



Note

If necessary, you can insert a parameter field with a right-click on the text box. You can assign a process and a file format to the parameter.

Note

Formatting of the text

If you use the clipboard to copy a text into an alarm or additional text, formatting may become invalid. You can delete the formatting with the command "Delete formatting". You can execute the command via the shortcut menu by right-clicking on the corresponding text.

Result

Alarm texts have been defined for the selected alarm.

6.2.9 Configuring info texts

You can write info texts for analog alarms and discrete alarms that can be displayed in runtime by using the "Info text setup" button of the alarm control.

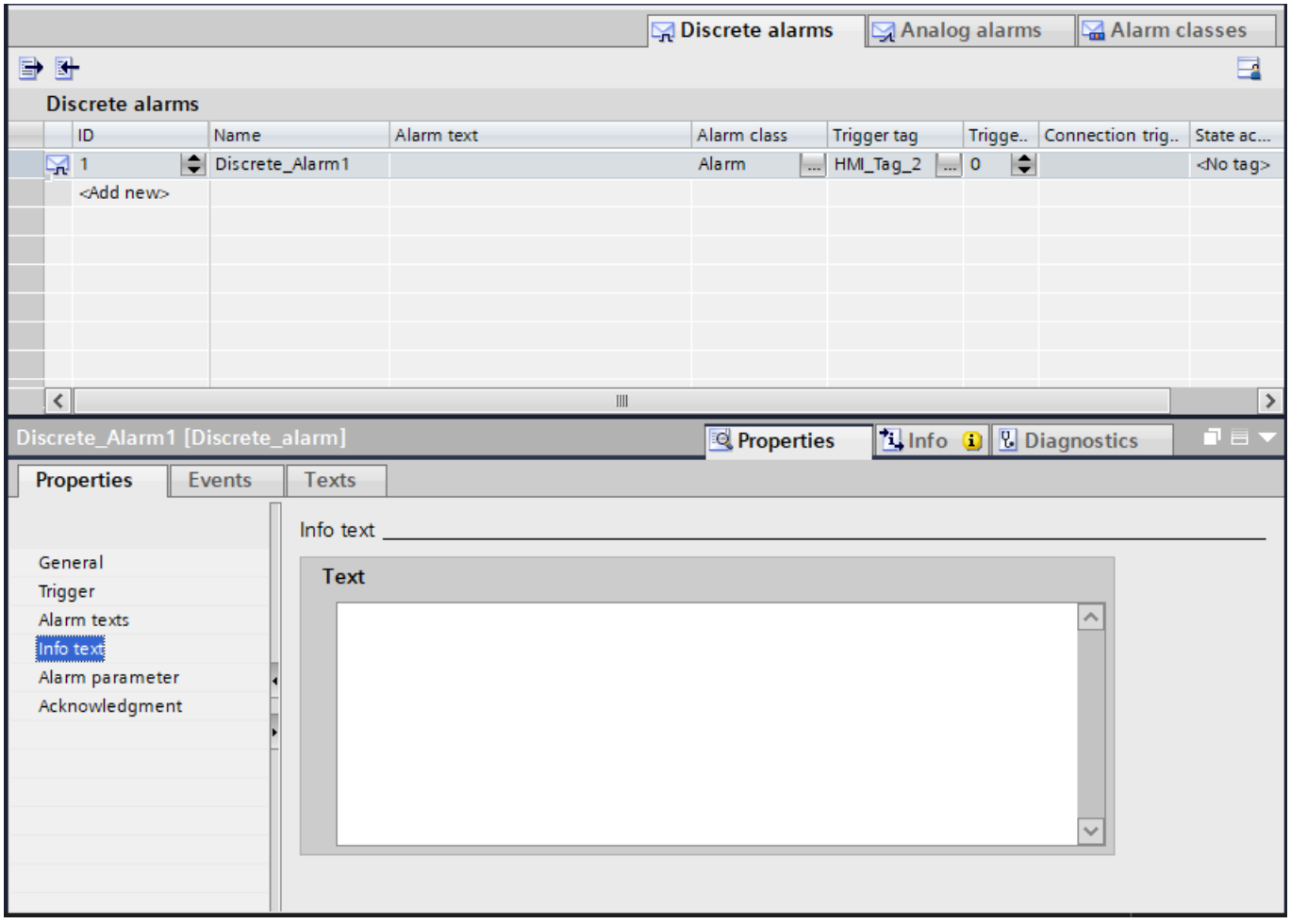
Requirement

- The "HMI alarms" editor is open.
- A discrete alarm has been created.

Procedure

To define alarm texts, follow these steps:

1. Select the analog or discrete alarm.
2. Click on "Info text" under "Properties" in the Inspector window.
   The "Info text" window opens.
3. Enter the info text.



Result

An info text has been defined for the selected alarm.

6.2.10 Parameter output in a discrete or analog alarm

Introduction

To display alarm parameters, insert an appropriate output field in a discrete or analog alarm. You can select the parameters configured in "Properties > Properties > Alarm parameters" for use as alarm parameters.

Requirement

- The "HMI alarms" editor is open.
- The discrete alarm or analog alarm is selected.

Procedure

To output a parameter in the alarm text, follow these steps:

1. Place the cursor at the required position in the alarm text.
2. To output an alarm parameter, select "Insert parameter field" from the shortcut menu.
   A dialog box opens.
3. Select the desired parameter.
4. Moreover, you can specify the following data for alarm parameters:
   - The tag that provides the parameter values.
     The tag configured for the parameter under "Properties > Properties > Alarm parameters" is entered by default. If you select a different tag, WinCC updates the parameter configuration in "Properties > Properties > Alarm parameters" accordingly.
   - Display type, text list, length, number of decimal places and alignment of the output field
   - To display leading zeros in the output field, enable "Leading zeros".
5. Confirm the dialog to save your entries.

6.2.11 Configuring optional parameters for discrete alarms and analog alarms

Setting the alarm context

In a large plant system, it makes sense to save information about the alarm origin such as the physical, geographical or logical grouping of plant units that is defined by the site. This helps users in identifying the causes of the alarm and the source of the fault.

You configure the information on alarm sources in the "Origin" field of the "Alarm context" area.

The "Area" field is a static field and contains information about the device.

Creating info texts for alarms

To configure an info text for the alarm and thus support users, follow these steps:

1. Select a discrete alarm or an analog alarm.
2. Select "Properties > Info text" in the Inspector window and enter the required text.
3. To insert a line break in the info text, press "Shift+Enter" at the corresponding text location.
4. To create multi-lingual info texts, enter the respective texts in the predefined project languages in the Inspector window and, if necessary, in the reference language.

Enabling parameters for a discrete or analog alarm

To output process values in an output field in the alarm text, assign tags to the parameter blocks. Proceed as follows:

1. Select the alarm.
2. In the Inspector window, click "Properties > Alarm parameters".
3. Select a tag for the alarm parameter.
4. You can enter multiple alarm parameters if required.
   Insert the activated process values as a selection box in an alarm text.

Note

You can configure up to 10 tags as alarm parameters for discrete alarms and analog alarms.

All available data types are supported.

See also

Configuring analog alarms (Page 738)

Configuring discrete alarms (Page 734)

Workflow for configuring alarms (Page 724)

Configuring multilingual alarm texts (Page 747)

Editing system events (Page 748)

Configuring alarm acknowledgment (Page 749)

Translating texts directly (Page 233)

Exporting project texts (Page 235)

Analog Alarms (Page 709)

Discrete alarms (Page 710)

6.2.12 Configuring multilingual alarm texts

Requirements

- The "HMI alarms" editor is open.
- An alarm has been created.

Procedure

1. Select one or more alarms for which you want to configure multilingual alarm texts.
2. You can view the alarm texts already configured in the set project languages under "Properties > Texts".
3. If available, enter the alarm texts in the required project languages.
   The alarm texts will then be displayed in the set runtime language in runtime.

Note

All alarm texts are managed together with other project texts under "Languages & Resources > Project texts".

If you cannot configure the project texts in multiple languages yourself, export them to an Excel file and have them translated. You can then import the texts to your project.

See also

Basics of project texts (Page 232)

Workflow for configuring alarms (Page 724)

Configuring discrete alarms (Page 734)

Configuring analog alarms (Page 738)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

Working with multiple languages (Page 231)

Importing project texts (Page 237)

Translating texts directly (Page 233)

Selecting the reference language and editing language (Page 229)

6.2.13 Editing system events

Basics

A system event indicates the system status and communication errors between the HMI device and system. System events are output in runtime in the configured alarm control. System events are output in the language currently set on your HMI device.

The time format (AM/PM or 24-hour format) is based on the selected language. If no translation of the alarm texts exists in this language, English is used as replacement and the corresponding time format is displayed.

Example of an alarm:

"Memory is full!"

Editing system events

You edit system events in the "HMI alarms" editor or under "Languages & Resources > Project texts". In the "Category" column you can recognize a system event by the name "HMI system event". You can export the system events together with the other texts under "Project texts" and have them translated.

System event parameters

System events can contain encrypted parameters. The parameters are of relevance when troubleshooting because they provide a reference to the source code of the runtime software. These parameters are output after the "Error code: text"

See also

Workflow for configuring alarms (Page 724)

Configuring discrete alarms (Page 734)

Configuring analog alarms (Page 738)

Configuring optional parameters for discrete alarms and analog alarms (Page 745)

System events (Page 712)

6.2.14 Filtering controller alarms via display classes

Introduction

Controller alarms are configured in STEP 7. Controller alarms are available in WinCC running in a STEP 7 environment.

If a PLC is connected to multiple HMI devices, the project engineer assigns display classes to the controller alarms in STEP7. The display classes determine the allocation to the HMI device. You can activate the display classes for your HMI device that are to be displayed on

it. In this case, only the controller alarms from this display class will be displayed on the HMI device. Up to 17 display classes are possible.

Note

WinCC only supports controller alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the controller.

Requirement

- The connection was established to the PLC.
- Alarms were configured in STEP 7.

Filtering controller alarms via display classes

To filter controller alarms by display classes, proceed as follows:

1. Click "Runtime settings > Alarms" in the project tree under your HMI device.
   One or several connections to a PLC are shown in "Contoller alarms".
2. Select the display classes whose controller alarms you want to display for the connection.

See also

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Configuring automatic update of controller alarms on the HMI device (Page 797)

Workflow for configuring alarms (Page 724)

User-defined controller alarms (Page 710)

Alarm system (Page 707)

6.2.15 Configuring alarm acknowledgment

Introduction

The alarm classes define how the alarms from an alarm class are to be acknowledged. When you assign an alarm to an alarm class, you define the state machine and the acknowledgment model for that alarm.

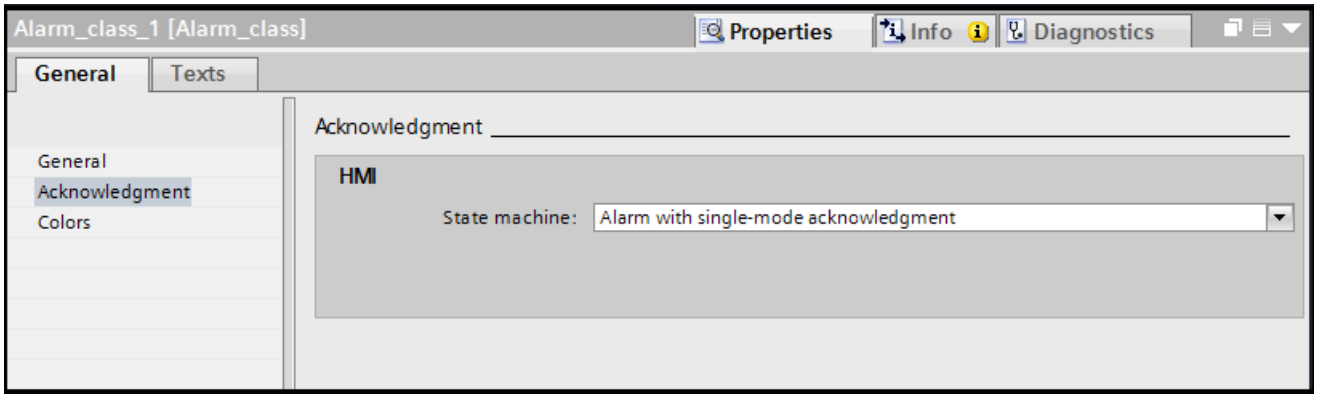
Requirements

- The "HMI alarms" editor is open.
- The required alarm class has been created.
- The required alarm has been created.

Procedure

To configure the acknowledgement of an alarm, follow these steps:

1. In the "HMI alarms" editor, click the "Alarm class" tab and select the alarm class.
2. You select the desired state machine under "Properties > General > Acknowledgment" in the Inspector window.



Note

The buttons relevant for acknowledgment, "Group acknowledgment", "Single acknowledgment" and "Single confirm", are enabled in the alarm view by default and can be operated in runtime.

Configuring the state machine of common alarm classes

The "Alarm class" tab also shows alarm classes that are linked to common alarm classes. The "General > Common alarm class" property is set for these alarm classes.

Alarm classes with such a link have one of the following state machines:

- Alarm with single-mode acknowledgment
- Alarm without acknowledgment

It is not possible to change the state machine in the "HMI alarms" editor. If necessary, change the state machine in the editor for common alarm classes.

See also

Acknowledgment model (Page 715)

Acknowledging alarms (Page 721)

Acknowledging alarms (Page 784)
Workflow for configuring alarms (Page 724)
Creating alarm classes (Page 726)
Configuring discrete alarms (Page 734)
Configuring analog alarms (Page 738)
Configuring optional parameters for discrete alarms and analog alarms (Page 745)
Using common alarm classes (Page 729)

6.3 Exporting and importing alarms

6.3.1 Exporting alarms

Introduction

WinCC makes an export function available for alarms.

Requirements

- The WinCC project for export is open.
- Alarms have been created in the project.
- The "HMI alarms" editor is open.

Exporting alarms

To export alarms from a WinCC project, follow the steps below:

1. Click the  button in the "Discrete alarms" or "Analog alarms" tab.
   The "Export HMI alarms" dialog box opens.
2. Click "..." and specify the file in which data is saved.
3. Specify whether you want to export "Discrete alarms" and/or "Analog alarms".
4. Click "Export".
   The export starts. When the export is complete, a message on completion of the export is displayed.
5. Confirm the message on completion of the export with "OK".

Result

The exported data has been written to an xlsx file. The xlsx file has been stored in the specified folder.

If you have only exported discrete alarms, the xlsx file has the worksheet "DiscreteAlarms". If you have only exported analog alarms, the xlsx file has the worksheets "AnalogAlarms" and "Limits". If you have exported discrete alarms and analog alarms, the xlsx file has the worksheets 'DiscreteAlarms", "AnalogAlarms" and "Limits".

Each alarm is in a separate row in the xlsx file.

Note

The list entries with the "FieldInfo" designation specify whether the alarm text contains dynamic parameters. The settings are separated by a semicolon ";".

6.3.2 Importing alarms

Introduction

WinCC makes an import function available for alarms. Alarms are identified by their alarm ID. An existing alarm is overwritten by the data from the import file if the alarm ID already exists in the project on import. A new alarm is created in the project if the alarm does not yet exist in the project on import.

Requirements

- An xlsx file with alarms has been created.
- The xlsx file has the same structure as an xlsx file that is created when alarms are exported.
- The IDs and names that were assigned for messages in the xlsx file are unique throughout the project.
- The WinCC project for import is open.
- The "HMI alarms" editor is open.

Importing alarms

To import alarms into a WinCC project, follow the steps below:

1. Click the  button in the "Discrete alarms" or "Analog alarms" tab.
   The "Import HMI alarms" dialog box opens.
2. Click "..." and select the file that you want to import.

3. Click "Import".
   The import starts. An xml log file is created on import. When the import is complete, a message on completion of the import is displayed.

   Note

   To open the created xml log file, click the link "Click here to view the log file". It is advisable to open the xml log file, especially if the import was completed with warnings.

4. Confirm the message on completion of the import with "OK".

6.4 Configuring an alarm control

6.4.1 Configuring an alarm control

Introduction

The alarm control is configured for a screen. Current or logged alarms are displayed in the alarm control in runtime. More than one alarm can be displayed simultaneously, depending on the configured size. Configure the criteria for alarm filtering.

You can also configure multiple alarm controls with different contents and in different screens.

Note

Text alignment in cells of the alarm control

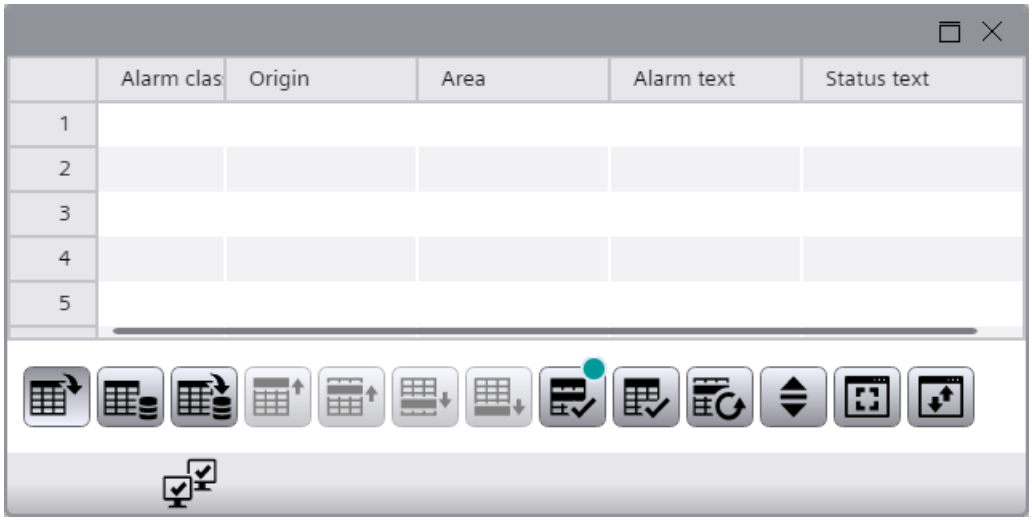
The "Spacing" and "Text trimming" properties only have an effect in the "Graphic and Texts", "Graphic or Text" and "Graphic" modes.

Requirement

- A screen is open.
- The "Toolbox" task card is open.

Procedure

1. Insert an "Alarm control" object from the "Tools" task card into the screen.



2. Go to "Properties" and set the required height, width and position of the alarm control.
3. Under "Properties > Miscellaneous > Alarm control" you can define the layout and color composition of the alarm control as well as the design of the header and the contents of the table grid.
4. To ensure that the most current alarm is always displayed and highlighted in the alarm control in runtime, activate the "Miscellaneous > Alarms - Show current" property.
   The visible area of the alarm control is moved in runtime if necessary. Users cannot select alarms individually or sort them by column. Alarms that have been filtered out of the alarm control are not displayed.
   If you configure the "Alarms - show current" button as visible and operable, users can pause and start this behavior in runtime as required. The alarm control always starts with the behavior configured via "Miscellaneous > Alarms - show current".
5. Under "Alarm source" you specify which alarms the alarm control displays in runtime by default.
   Depending on your task or the requirements in your company, you can select from the following display options:
   - "Not configured": The alarm control does not show any alarms.
   - "Pending alarms": The alarm control shows the currently pending alarms.
   - "Logged alarms": The alarm control shows the logged alarms.
   - "Logged alarms updated": The alarm control shows the logged alarms that are updated at specified intervals.
   - "Alarm definition": The alarm control shows all alarms configured in the engineering system, regardless of whether or not they have occurred.

   Depending on your selection, the alarm control display already changes in the engineering system. The buttons relevant for the settings are shown as being active, while buttons that are not relevant are grayed out. These settings are applied for runtime.

6. Define which alarms are displayed in runtime by default in the alarm lists for pending alarms and for defined alarms.
   - In the selection list under "Miscellaneous > Current alarms", select which alarms are displayed as pending alarms.
   - In the selection list under "Miscellaneous > Displayed alarms", select which alarms are displayed as defined alarms.

   Depending on your task or the requirements in your company, you select one or more display options depending on the status of the alarms:
   - "None": The alarm control shows all alarms.
   - "Not suppressed": The alarm control only shows the non-suppressed alarms.
   - "Locked": The alarm view only shows the locked alarms.
   - "Suppressed by design": The alarm view only shows the alarms suppressed by design.
   - "Shelved": The alarm control only shows the shelved alarms.

   Your selection is displayed by default in the alarm control when you start runtime.

   Note

   If you do not make a selection, the alarm control shows all alarms.

   Note

   You can change the display at any time in runtime even if you have selected a different display option in the engineering system under "Alarm source" or "Active alarms".

7. If necessary, select the authorization needed to operate the alarm control in runtime.
8. Under time zone you set the desired time zone by entering a decimal value for the time zone.
   - "0" and positive numerical values: The values correspond to the index values of the Microsoft time zones.
   - "-1": The local time zone of the device

   Note

   In runtime you also have the option of setting the time zone via a selection list.

Result

Alarms of various alarm classes are output in the alarm control during runtime. To change the control in runtime, click the configured buttons on the alarm control toolbar.

Using alarm colors

To display the colors configured for an alarm in the alarm control, proceed as follows in the engineering system:

1. Activate the property "Format > Use alarm color" in the properties of the alarm control.
2. Activate the "Use alarm color" property under "Alarm control > Columns" for each column that is to use the configured alarm color.

See also

Configuring the toolbar (Page 757)

Configuring filters in the alarm control (Page 762)

Configuring alarm export (Page 764)

Configuring the printing of alarms (Page 764)

Show logged alarms (Page 765)

Defining the output format (Page 364)

Configuring columns and sorting (Page 760)

Alarm control (Page 309)

Configuring an alarm control for plant objects (Page 7084)

Configuring state texts of alarms (Page 733)

Display messages from participating devices (Page 7583)

Configuring reordering of the columns (Page 392)

6.4.2 Display all information about an alarm

Introduction

When configuring an alarm view, you have the possibility to provide the operator with a pop-up in Runtime that shows further information about an alarm selected in Runtime.

The following system function is available for this purpose:
Alarm.GetSelectedAlarmAttributes() (Page 1480).

Requirement

- The alarm view has been configured.

Procedure

1. In the Inspector window, select the "Selection changed" event.
2. Call the "GetSelectedAlarmAttributes" function in a script.
   All attributes of the alarm are read out.
3. Display the attributes in Runtime in a pop-up.

The output of the attributes can also be triggered by a button.

A snippet "Get all alarm properties of the selected alarm from the alarm control" is available under "HMI Runtime > Screen" in the script editor.

Result

The "Show current" button (Autoscroll) must be switched off.

The operator selects an entry in Runtime in the alarm view.

A pop-up with the attributes of the alarm is displayed.

The function also outputs attributes for archived alarms and alarms of the alarm statistics.

6.4.3 Configuring the toolbar

Introduction

You operate the alarm control in runtime using the buttons in the toolbar. During configuration, you define the contents of the toolbar.

The following buttons are visible in the alarm control by default:

- "Show active alarms"
- "Show logged alarms"
- "Show and update logged alarms"
- "Alarm statistics - view"
- "First line"
- "Previous line"
- "Next line"
- "Last line"
- "Single acknowledgment"
- "Group acknowledgment"
- "Single confirm"
- "Alarms - show current"
- "Alarm statistics - configuration"

- "Selection display"
- "Sorting setup"

The following buttons are not visible in the alarm view by default and must be made visible via "Properties > Miscellaneous > Toolbar > Elements". To these additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

- "Show defined alarms"
- "Alarm annunciator"
- "Move to next acknowledgeable alarm"
- "Previous page"
- "Next page"
- "Info text setup"
- "Comments setup"
- "Statistics setup"
- "Disable alarm"
- "Enable alarm"
- "Shelve alarm"
- "Unshelve alarm"
- "Copy lines"
- "Time base setup"
- "Display options setup"
- "Disabled alarms setup"
- "Export"
- "Select context"

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Configuring the toolbar

1. Configure the general properties of the toolbar, such as alignment or background color, in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar".
2. In the Inspector window, enable the buttons you need in runtime, e.g. "Export" or "Print", under "Properties > Properties > Miscellaneous > Toolbar > Elements".
3. Configure the display of the respective button, e.g. background color, border, and maximum and minimum size.

4. If needed, you can define a tooltip for the buttons.
5. If a button is not to be operated in runtime, deselect "Operator control - allow".
   You can reactivate a deactivated a button using a script in runtime, for example.

Note

The order and functionality of the buttons are defined in the system and cannot be changed.

Custom ID

The custom ID can be used if you want to specifically access a certain button with an ID to be set via scripting.

You set this property under "Properties > Properties > Miscellaneous > Toolbar > Elements > Button".

Start behavior for "Alarms - Show current"

The "Alarms - Show Current" button starts with the behavior configured in "Properties > Miscellaneous > Show Current".

If you configure the button as operable, users can change the behavior in runtime. By clicking the button they pause the display of the current alarms or start it again as required.

See also

Configuring an alarm control (Page 753)

Configuring columns and sorting (Page 760)

Configuring filters in the alarm control (Page 762)

Configuring alarm export (Page 764)

Configuring the printing of alarms (Page 764)

Show logged alarms (Page 765)

6.4.4 Configuring the information bar

Introduction

The information bar displays status messages of the alarm control. During configuration, you configure the content of the information bar.

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Configuring the information bar

1. In the Inspector window, configure the general properties of the Information bar such as the font or the background color under "Properties > Properties > Miscellaneous > Status bar".
2. In the Inspector window, select the elements you need in runtime, such as date, time, connection status, etc. under "Properties > Properties > Miscellaneous > Information bar > Elements".
3. To adjust the size of an element in the information bar, select "User-defined".
4. Enter the width and height in pixels.
5. To set the order of the elements, select the element in the list and move it to the desired position.

Custom ID

The custom ID can be used if you want to specifically access a certain button with an ID to be set via scripting.

You set this property under "Properties > Properties > Miscellaneous > Toolbar > Elements > Button".

6.4.5 Configuring columns and sorting

Introduction

You configure the order in which the columns of the alarm control are displayed in runtime.

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Configuring columns

1. Click "Properties > Alarm control > Columns" in the Inspector window.
2. Enable the "Visibility" property for the relevant columns.

3. Under "Alarm text block" select the content that is to be displayed in the column, e.g. "Alarm class".
4. Under "Alarm column [n] > Header > Text", enter the desired column name that is to be displayed in the alarm control.

Note

For the column names to be configured as multilingual, you must enter the name of the column under "Alarm column [n] > Header > Text".

You will then see the configured text in the Inspector window under "Texts" and can store additional languages.

If you only enter the name under "Alarm column [n] > Name", multilingual configuration is not possible.

Configuring the sorting

To sort alarms in the alarm control by column, follow these steps:

1. Select "Properties > Alarm control" > Allow sorting" so that sorting is generally possible in the alarm control in runtime.
2. Under "Properties > Alarm control > Columns" open the alarm column by which you want to initially sort the alarms, e.g. the "Priority" column.
3. Select the sorting order "1".
4. Select the desired sorting direction, e.g. "Ascending".
   - The number "1" with the arrow pointing upwards for ascending sort order is displayed in runtime in the column with sorting order "1".
   - If the sorting order "Ascending" is enabled in the alarm control, each click in the column header toggles the sorting between the ascending and descending mode.
5. To allow sorting for this column, enable "Alarm column [n] > Allow sorting".

Note

You can configure any sorting order.

If the "Show recent" property was selected under "Properties", the latest alarms are always shown first.

See also

Configuring an alarm control (Page 753)

Configuring the toolbar (Page 757)

Configuring filters in the alarm control (Page 762)

Configuring alarm export (Page 764)

Configuring the printing of alarms (Page 764)

Show logged alarms (Page 765)

6.4.6 Configuring filters in the alarm control

Introduction

You can filter the display of alarms in the alarm control. You configure a static value, a tag or a script for the filter. You can configure this function in the alarm control in the "Screens" editor. To filter the alarms in Runtime, click "Selection display" in Runtime.

You can filter by all parameters, such as ID, name, alarm class, priority, etc.

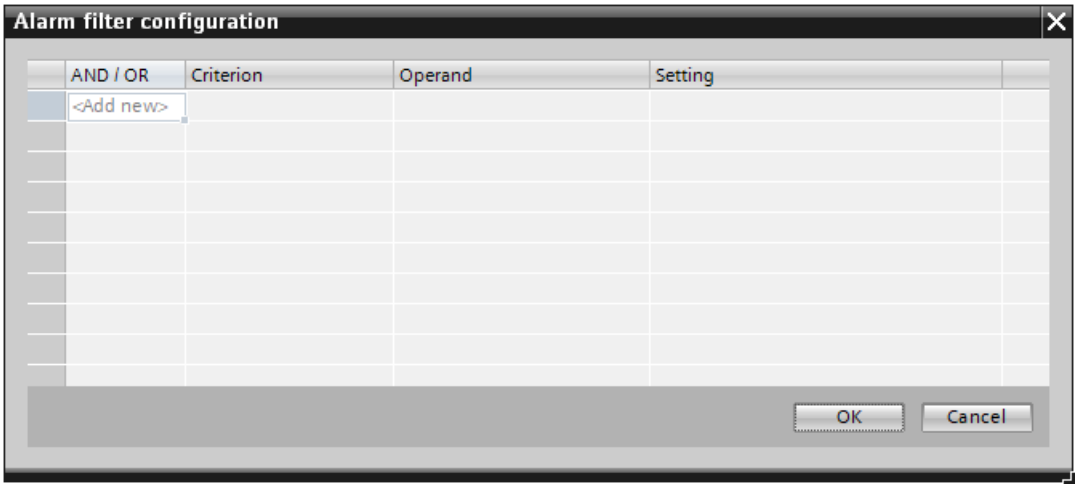
This operating instruction introduces the process for configuring a filter using an example.

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Procedure

1. In the Inspector window under "Properties > Filter", click on the "..." button in the "Static value" column.
   The "Alarm filter configuration" dialog box opens.



2. Click "<Add>" in the "AND/OR" column to create a filter.
3. In the "Criterion" column, open the selection list and select the entry "Alarm class".
4. In the "Operand" column, open the selection list and select the entry "Equal to".
5. Enter the value "Alarm" in the field of the "Setting" column.
6. Click the "OK" button.

Note

The following characters cannot be used in the Settings column:

- Quotes
- Single quotation marks

Tips for an efficient procedure

You can also create filter criteria directly in runtime and use them as filters. To operate the filter in runtime, enable the "Visibility" property under "Properties > Toolbar > Elements > Control bar button Selection display [26]".

Examples of alarm filters

Example	Criterion	Condition	Setting	Description
Filter by alarm class	Alarm class	=	Alarm	Displays the alarms that are members of the "Alarm" alarm class.
Filter by priority	Priority	Greater than	4	Displays the alarms with a priority greater than 4.
Filter by alarm text	Alarm text	Contains	Motor_12	Displays the alarms that contain the alarm text "Motor_12".
Filter by time	Last modification	Less than or equal to	Wednesday, 15 April 2020 17:00	Displays the alarms that occurred after 5 pm (17:00) on 15 April 2020.

Filter by time

When filtering by time, the start and stop values are not adjusted automatically when the time base of the alarm control is changed.

Example

At a PC location with time zone "UTC + 1h", the alarm control has the "Local time zone" time base. If you filter for the time 10:00 to 11:00 and then change the time base to "UTC", you need to change the start value and stop value of the filter to 9:00 and 10:00 to display the same alarms as before.

See also

Filtering alarms in runtime (Page 778)

Configuring an alarm control (Page 753)

Configuring the toolbar (Page 757)

Configuring columns and sorting (Page 760)

Configuring alarm export (Page 764)

Configuring the printing of alarms (Page 764)

Show logged alarms (Page 765)

Alarm control (Page 309)

6.4.7 Configuring alarm export

Introduction

To export alarms to a "*.csv" file in runtime, click on the "Export" button in the alarm control. You configure the "Export" button in the alarm control in the "Screens" editor.

Requirement

- The screen with the configured alarm control is open.
- The Inspector window is open.

Procedure

To configure the export of alarms, proceed as follows:

1. Select the alarm control and enable the "Visibility" property in the Inspector window under "Properties > Toolbar > Elements > Export button [29]".

You define the export settings such as the file name, the scope of the export and the format in runtime in the "Export data" dialog.

See also

Configuring an alarm control (Page 753)
Configuring the toolbar (Page 757)
Configuring columns and sorting (Page 760)
Configuring filters in the alarm control (Page 762)
Configuring the printing of alarms (Page 764)
Alarm control (Page 309)

6.4.8 Configuring the printing of alarms

Introduction

Click "Print" in the alarm control to print alarms in runtime. You configure the "Print" button in the alarm control in the "Screens" editor.

Requirement

- The screen with the configured alarm control is open.
- The Inspector window is open.

Procedure

To configure the printing of alarms, follow these steps:

1. Select the alarm control and enable the "Visibility" property in the Inspector window under "Properties > Toolbar > Elements > Print button [28]".

See also

Configuring an alarm control (Page 753)

Configuring the toolbar (Page 757)

Configuring columns and sorting (Page 760)

Configuring filters in the alarm control (Page 762)

Configuring alarm export (Page 764)

Show logged alarms (Page 765)

Alarm control (Page 309)

6.4.9 Show logged alarms

Overview

When an alarm log is created, an alarm control also shows logged alarms in runtime.

The buttons relevant for logging, "Show logged alarms" and "Show and update logged alarms", are activated in the alarm control by default and can be operated in Runtime.

You show logged alarms in runtime using these buttons.

See also

Basics of alarm logging (Page 769)

Displaying logged alarms in runtime (Page 779)

Configuring an alarm control (Page 753)

Configuring the toolbar (Page 757)

Configuring columns and sorting (Page 760)

Configuring filters in the alarm control (Page 762)

Configuring the printing of alarms (Page 764)

Configuring the display of security events (Page 795)

Alarm control (Page 309)

6.4.10 Configuring alarm statistics

Introduction

The alarm statistics in the alarm control shows statistical calculations for logged alarms. The alarm statistics also shows a selection of the configured alarm blocks. If there is dynamic content, the alarm blocks show the data of the alarm last arrived. The columns of the alarm statistics can be arranged individually.

The alarm statistics shows the following statistical calculation types:

- Frequency of an alarm
  The system counts the number of occurrences of an alarm with "active" status in the log. If the alarm number is not found, this alarm number is not included in the statistics.
- Total display time of an alarm in seconds
  You can calculate the following time periods between alarm states:
  - Sum active active
  - Sum active inactive
  - Sum active acknowledged
- Average display time of an alarm in seconds
  You can calculate the following time periods between alarm states:
  - Average active active
  - Average active inactive
  - Average active acknowledged

There is a column available in the alarm statistics for each type of calculation.

The calculation of the time of acknowledgment includes the "acknowledged" alarm status. This "acknowledged" alarm state includes the acknowledgment by the controller.

Requirement

- The alarm control is selected in the screen.
- The Inspector window is open.

Setting up columns of the alarm statistics

To set up the alarm statistics columns, proceed as follows:

1. In the Inspector window, click "Properties > Miscellaneous > Alarm statistic view > Columns".
2. Select the "Visibility" option for the desired columns.

Settings for calculating the alarm statistics

You can configure the following options:

Setting	Description
Start time	Start time for the calculation.
Time range start	• Now The current time is displayed as the start time of the calculation. • Fixed The start time of the calculation can be changed as required.
Time range base	Unit of time for the calculation. The following settings are available: • Undefined The default time unit "Minute" is used with this setting. • Millisecond • Second • Minute • Hour • Day • Month • Year
Time range factor	The time range factor depends on the "Time range base" setting. For example, if the number 4 is set for the time range factor and "Minutes" is set for the time range base, all alarms that are logged within this period will be evaluated.

To configure the settings for the alarm statistics, proceed as follows:

1. In the Inspector window, click "Properties > Miscellaneous > Alarm statistics settings".
2. Specify the desired properties.

6.4.11 Configuring the display of system diagnostic alarms

Introduction

System diagnostic alarms are installed with STEP 7 and are used for monitoring states and events of a controller. To display system diagnostic alarms in an alarm control in runtime, configure first in STEP 7 and then in WinCC.

Note

WinCC only supports system diagnostic alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports system diagnostic alarms that are updated by the central alarm management in the controller.

Requirement

- There is an HMI connection between the HMI Device and a SIMATIC S7-1500 controller (as of firmware version 2.0).

Configuring the display of system diagnostic alarms in STEP 7

To configure the display of system diagnostic alarms in runtime in STEP 7, proceed as follows:

1. Open the "Device configuration" of the controller in the project tree.
2. In the "Device view" tab, select the CPU on the rack.
3. Select "Properties > General > System diagnostics" in the Inspector window.
   You will see that the option "Select system diagnostics for this device" is selected and cannot be cleared. Because the system diagnostics of the controller is always enabled.
4. Activate the option "Central alarm management in the PLC" in the Inspector window under "Properties > General > PLC alarms".
   The automatic update of system diagnostic alarms on the HMI device is enabled in the controller.
5. Open the "Common data" folder in the project tree and double-click "System diagnostic settings".
   The system diagnostic settings are opened. You can see the predefined categories of the system diagnostic alarms in the table under "Category":
   - "Error"
   - "Maintenance demanded"
   - "Maintenance required"
   - "About"
6. In the table under "Category", select the alarm categories that are to be displayed in the alarm control in runtime.
7. In the table under "Alarm class", assign common alarm classes to the alarm classes.
8. Right-click the controller in the project tree and select "Compile > Hardware (rebuild all)" in the shortcut menu.

Configuring the display of system diagnostic alarms in WinCC

To configure the display of system diagnostic alarms in WinCC in runtime, proceed as follows:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Select the option "Automatic update" under "Alarms > Controller alarms".
   The automatic update of system diagnostic alarms on the HMI device is enabled in the HMI device.
3. Select the option "System diagnostics" under "Alarms > Controller alarms".
   The display of system diagnostic alarms is enabled in runtime.
4. Configure an alarm control.

Result

The alarm control displays of the system diagnostic alarms of the controller in runtime.

See also

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Configuring automatic update of controller alarms on the HMI device (Page 797)

System-defined controller alarms (Page 712)

Workflow for configuring alarms (Page 724)

6.5 Logging alarms

6.5.1 Basics of alarm logging

Introduction

An alarm log is used to log alarms that occur in the monitored process. You can use alarm logging to analyze error states and to document the process. When you analyze the logged alarms, you can extract important business and technical information regarding the operational state of the plant.

Alarms of connected and appropriately configured PLCs are also logged and made available in all configured languages.

How it works

Each alarm is assigned to an alarm class. To ensure clarity with large amounts of data, alarm classes can be prioritized and configured differently. Under "HMI alarms", you assign logs to the alarm classes.

Alarm logs are created by the system in runtime. For example, when a fault or a limit violation occurs, the corresponding alarm you configured in the "HMI alarms" editor is output in runtime. Each alarm event is logged, for example, the status change of the alarm from "incoming" to "acknowledged".

Content of the alarm log

Alarms and their properties in all configured languages are saved in the alarm logs. The time stamp of a logged alarm is always specified in standard UTC format (Universal Time Coordinated).

Displaying logged data

You display the logged data in runtime in the alarm control. To display logged data, use the "Show logged alarms" button in the alarm control.

See also

Show logged alarms (Page 765)

Creating a data log and an alarm log (Page 848)

How it works (Page 839)

Log basics (Page 837)

Storage locations of logs (Page 843)

Editing log contents with scripts and system functions (Page 850)

Size of a log entry in the alarm log (Page 770)

6.5.2 Size of a log entry in the alarm log

The size of a log entry depends on the following factors:

- Number of configured languages
  You configure the languages to be logged in the Runtime settings of the HMI device. When more than one language is defined, memory is required for each additional language to store the language entry.
- Alarm texts and additional texts
  - Length of the alarm texts and additional texts
  - Memory requirement of individual characters
    Symbols and complex characters have greater memory requirements.
- Database type used

Calculation

The size of a log entry is calculated as follows:

Basic entry without alarm text + memory requirement of all texts + (number of configured languages-1)*memory requirement of an additional language entry

The text lengths and the memory requirements of the individual characters depend on the respective language. The following formula can be used to easily estimate the memory requirement of all texts:

Number of configured languages*total length of the alarm text and the additional texts*memory requirement of a character

The individual values depend on the database type used.

	SQLite	Microsoft SQL
Additional memory requirement per segment	-	Approx. 3.5 MB
Basic entry without alarm texts	Approx. 300 bytes	Approx. 2000 bytes
Memory requirement of an additional language entry	Approx. 100 bytes	Approx. 200 bytes
Memory requirements of a character	At least 1 byte	At least 2 bytes

See also

How it works (Page 839)

Multilingual logging of alarms (Page 772)

Basics of alarm logging (Page 769)

6.5.3 Assign alarm class

Introduction

You assign one alarm log each to predefined and user-defined alarm classes. In this way, you can categorize alarms in different logs.

Note

Assign alarm class

If you do not assign an alarm class to an alarm log, you will receive an error message when you compile the project.

Assign at least one alarm class to each alarm log.

Requirement

- An alarm log has been created in the "Logs" editor.

Procedure

1. Double-click on the "HMI alarms" entry in the project tree.
   The "HMI alarms" editor is opened.
2. Select the "Alarm classes" tab.
   The list of alarm classes is displayed.
3. Double-click on an entry in the "Log" column.
   The selection button is displayed.
4. Select an alarm log.

See also

Basics of alarm logging (Page 769)

Creating a data log and an alarm log (Page 848)

6.5.4 Multilingual logging of alarms

Introduction

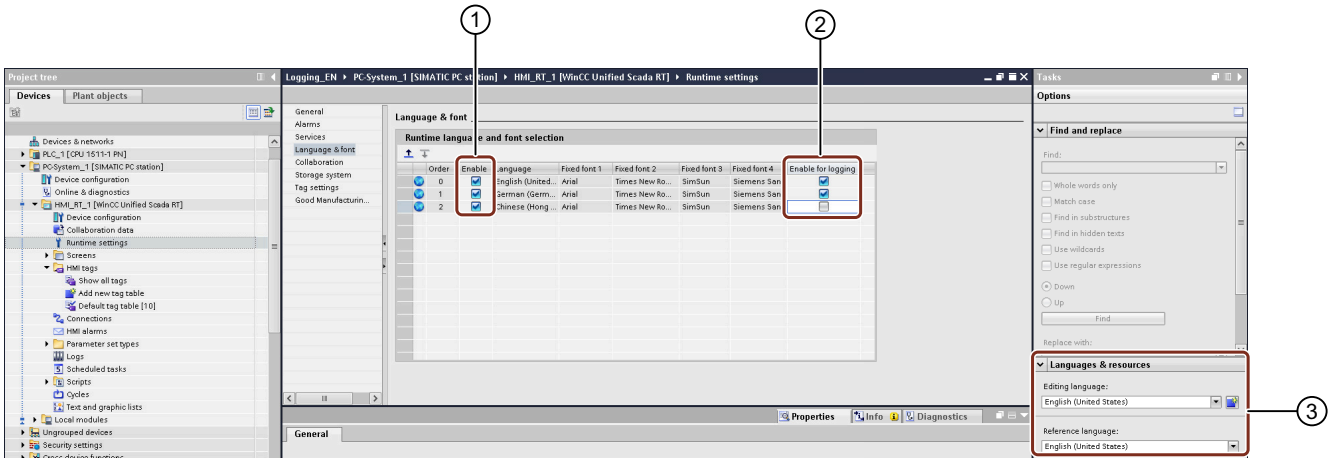
You can log alarm texts in different languages. To do this, select the option in the runtime settings of the HMI device.

Requirement

- The languages in which the alarm texts are to be logged are created as project languages.

Procedure

1. Open "Runtime settings" in the project tree below the HMI device.
2. Select "Language & Font".
3. To activate a language for runtime, select the check box in the "Activate" column.
   This option must be activated to log alarm texts in the corresponding language.
4. To log the alarm texts in a language, select the check box in the "Enable for logging" column.



① Activate available languages in Runtime
② Enabling languages for logging
③ Adding project languages

See also

Basics of alarm logging (Page 769)

Size of a log entry in the alarm log (Page 770)

6.6 Displaying and using alarms

6.6.1 Operating the alarm control and displaying it in runtime

Introduction

In runtime, the "Alarm control" object displays alarms that occur during the process in a plant. Alarms indicate events and states on the HMI device which have occurred in the system, in the process or on the HMI device itself. A state is reported when it occurs. Use the alarm control functions to work with incoming alarms. An alarm could trigger one of the following alarm events:

- Incoming
- Outgoing
- Acknowledge

The configuration engineer defines which alarms must be acknowledged by the operator.

Alarm control

The alarm control shows selected alarms or alarm events from the alarm buffer or alarm log. Whether alarm events have to be acknowledged or not is specified in your configuration. You can configure the order in which the alarms are displayed. At the first position, the current, or the oldest alarm will be displayed.

In the alarm control, you have the option to display various lists that filter and sort according to specific properties. To display the alarm lists in the alarm control and switch the alarm control in runtime, click the associated button in the alarm control toolbar.

Button	Description	Description
	Show active alarms	Shows the pending alarms.
	Show logged alarms	Shows the logged alarms. The view is not updated immediately when new incoming alarms occur.
	Show and update logged alarms	Shows the logged alarms. The display is updated immediately when new incoming alarms occur.
	Show defined alarms	Shows the alarms configured in the engineering system.

Note

The maximum number of alarms that can be displayed in an alarm control in runtime depends on the selected view. Note the information in the performance features.

Roles and authorizations

When working with the alarm control, you can assign roles and authorizations that determine who can configure and operate in the engineering system and runtime.

In the Engineering System under "Security Settings > Settings" you have the option to set the user name and password for the open project with which the project is now protected with "Protect this project". Under "Security settings > Users and roles" you create users under the "Users" tab, to whom you can assign roles as well as rights. In addition to the predefined roles, you can also define additional roles under the "Roles" tab and then assign them to a user.

Operator control with the mouse

Selecting and controlling alarms

- Click on an alarm.
- Click on a button in the toolbar.

The function of the button is applied to the alarm.

Reordering columns

You have the option of changing the column order configured in the engineering system. You can find additional information in the section Rearranging columns in runtime (Page 393).

Sorting alarms by column

You have the option of sorting the alarms by column. You can find additional information in the section Sorting alarms in runtime (Page 775).

Operator control with keyboard

Press <Shift + Enter> until the alarm control has the focus. Then select the alarm to be processed and control it using the toolbar.

Use the following keys:

Keyboard shortcut	Function
PAGE UP	Navigates up the alarm control row by row.
PAGE DOWN	Navigates down the alarm control row by row.
SHIFT + PAGE UP	View the alarm control column by column to the left.
SHIFT + PAGE DOWN	Navigates the alarm control column by column to the right.
CTRL + UP	Sets the selection as the start of row.
CTRL + DOWN	Sets the selection as end of row.
CTRL + RIGHT	Sets the selection as start of column.
CTRL + LEFT	Sets the selection as end of column.
HOME	Sets the selection as the start of row.
END	Sets the selection as end of row.

See also

Printing alarms in runtime (Page 793)

Filtering alarms in runtime (Page 778)

Displaying logged alarms in runtime (Page 779)

Acknowledging alarms (Page 784)

Group acknowledgement of alarms (Page 786)

Exporting alarms (Page 787)

Shelving alarms (Page 788)

Lock alarms (Page 791)

6.6.2 Sorting alarms in runtime

Introduction

In runtime, you can sort the alarms in the alarm control by column header.

Sorting alarms:

- In descending order by date, time, and alarm number. The most recent alarm is displayed at the top.
- Ascending:
  The sequence for ascending order is:
  - Incoming
  - Incoming/acknowledged
  - Incoming/acknowledged/outgoing
  - Incoming/outgoing/acknowledged
  - Shelved
  - Suppressed
- No sorting
  When the alarm control is sorted by columns, an arrow and a number are shown on the right in the column header. The arrow indicates the ascending or descending sort order. The number beside the arrow indicates the sort order of the column headers.

Standard sorting of alarms

- By priority
  If the "Priority" column is defined in the alarm view, sorting is based on alarm priority. As a result, in a single-line alarm control, only the top-priority alarm appears in the alarm window. A lower-priority alarm will not be displayed, even if it is has a more recent date. The alarms are displayed in chronological order.
- If one of the following columns in the alarm view is configured, sorting on a Unified Comfort Panel is according to the specified order.

1. Priority
2. Modification time
3. Raise time
4. Alarm state

If none of the columns are configured, sorting occurs according to the "Time" column.

Requirement

- In the Engineering System, "Allow sorting" is enabled in the alarm control for the respective columns.
- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the Engineering System:

	Sorting setup
	Show recent

- "Show recent" is paused in runtime.

Sort alarms by clicking on the column header

To sort alarms in the alarm control by column header, proceed as follows:

1. In runtime, click the column header of the alarm control you want to sort by.
   An arrow and a number are displayed in the column representing the order and method of sorting.

Sorting alarms via button

To sort alarms in the alarm control by "Sorting setup" button, proceed as follows:

1. In runtime, click the "Sorting setup" button in the alarm control.
   The window for sorting opens.
2. Use the drop-down list to select a column and define whether you want to sort in ascending or descending order.
3. Apply your criteria via the "OK" button.
   The alarm control is sorted according to your criteria. The current sorting of the columns is indicated by an arrow and a number.

Note

To ensure that changes to existing sort criteria in the button are applied without errors, it is recommended that you first reset the changes using the "Clear sort criteria" button.

See also

Filtering alarms in runtime (Page 778)

Printing alarms in runtime (Page 793)

Operating the alarm control and displaying it in runtime (Page 773)

Displaying logged alarms in runtime (Page 779)

Acknowledging alarms (Page 784)

Group acknowledgement of alarms (Page 786)
Exporting alarms (Page 787)
Shelving alarms (Page 788)
Lock alarms (Page 791)

6.6.3 Filtering alarms in runtime

Introduction

In runtime, you can use criteria to define which alarms you want to display in the alarm control. In the example below, only alarms that contain the alarm text "Motor on" are displayed. You use the "Selection display" button in the alarm control for this purpose.

Requirement

- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Selection display

Procedure

To filter alarms in the alarm control, proceed as follows:

1. Click "Selection display" in Runtime.
   The "Selection" dialog opens.
2. Under "Criterion" select the criterion "Alarm text".
3. Enter the alarm text "Motor on" in the "Settings" column.

Note

The following characters cannot be used in the Settings column:

- Quotes
- Single quotation marks

Result

The alarm control only shows those alarms that contain the words "Motor on" in the alarm text. If necessary, define additional filter criteria by selecting the required condition in the "AND/OR" column and the respective criterion in the "Criterion" column.

See also

Configuring filters in the alarm control (Page 762)

Sorting alarms in runtime (Page 775)

Displaying logged alarms in runtime (Page 779)

Printing alarms in runtime (Page 793)

Operating the alarm control and displaying it in runtime (Page 773)

Acknowledging alarms (Page 784)

Group acknowledgement of alarms (Page 786)

Exporting alarms (Page 787)

Shelving alarms (Page 788)

Lock alarms (Page 791)

Alarm control (Page 309)

6.6.4 Displaying logged alarms in runtime

Introduction

You can use the "Show logged alarms" and "Show and update logged alarms" alarm lists to display logged alarms.

Requirements

- An alarm log has been configured.
- All the archived data that you intend to display in runtime must be stored locally on the archive server. The alarm log does not allow access to backup files held elsewhere, such as on another computer in the network.
- The "Visibility" and "Allow operator control" settings are activated for the following alarm view buttons in the engineering system:

	Show logged alarms
	Show and update logged alarms

Procedure

1. To display only logged alarms, click on the "Show logged alarms" buttons in the alarm view:

   The alarm view shows the logged alarms. The view is not updated immediately when new incoming alarms occur.
   Each page shows a maximum of 1 000 alarms. Use the "Previous page" and "Next page" buttons for scrolling.

2. To show logged and current alarms, click on the "Show and update logged alarms" button in the alarm view.

   The alarm view shows the logged alarms. The view is updated immediately when new active alarms occur.
   The alarm view shows a maximum of 100 alarms.

Limitation for the "Show logged alarms" alarm list

For log alarms with identical time stamps, log alarms may, in rare cases, be skipped when scrolling forward and backward.

To display the skipped alarms, scroll again in the opposite direction.

Example

- The alarm log contains several thousand log alarms. Ten alarms in the log have an identical time stamp. The first five of these alarms are displayed at the end of the current page.
  The alarm view is sorted in ascending order by time stamp.
- Click on "Next page".
  You see the next 1 000 alarms whose time stamp is higher than the time stamp of the last alarm displayed on the previous page.
  The remaining five alarms with identical time stamps are skipped when changing to the next page.
- Click on "Previous page".
  You can now see all ten alarms with identical time stamps, as well as the next 990 alarms with a lower time stamp.

Note

If there are more than 1 000 log alarms with identical time stamps, not all the skipped alarms can be displayed by scrolling in the opposite direction.

See also

Show logged alarms (Page 765)
Filtering alarms in runtime (Page 778)
Acknowledging alarms (Page 784)
Printing alarms in runtime (Page 793)
Operating the alarm control and displaying it in runtime (Page 773)
Sorting alarms in runtime (Page 775)
Group acknowledgement of alarms (Page 786)
Exporting alarms (Page 787)
Shelving alarms (Page 788)
Lock alarms (Page 791)
Configuring the display of security events (Page 795)
Alarm control (Page 309)

6.6.5 Displaying alarm statistics

Introduction

The alarm statistics represent statistical calculations of logged alarms.

You can use a button in the alarm control to export the alarm statistics to an Excel file.

Note
Filter

A filter set in the alarm control is not effective in the alarm statistics.

Note
Display options

Display options selected via the "Display options setup" button in the alarm control are not effective in the alarm statistics.

Requirement

- Alarms are logged.
- For the following button of the alarm control, the "Visibility" and "Allow operator control" are enabled in the engineering system:

	Alarm statistics

Procedure

To display the alarm statistics in Runtime, proceed as follows:

1. Click the "Alarm statistics" button in the alarm control.

Result

The alarms to be displayed in the alarm statistics are specified in the engineering system. Depending on the engineering system, the following columns are displayed:

Column	Description
Number	Configured number of the alarm.
Frequency	Frequency of an alarm. The system counts the number of occurrences of an alarm with "active" status in the log. If the alarm number is not found, this alarm number is not included in the statistics.
Sum active active	Total display time of an alarm in seconds. The time period between the alarm states "active" and "active" is calculated.
Sum active inactive	Total display time of an alarm in seconds. The time period between the alarm states "active" and "inactive" is calculated.
Sum active acknowledged	Total display time of an alarm in seconds. The time period between the alarm states "active" and "acknowledged" is calculated.
Average active active	Average display time of an alarm in seconds. The time period between the alarm states "active" and "active" is calculated.
Average active inactive	Average display time of an alarm in seconds. The time period between the alarm states "active" and "inactive" is calculated.
Average active acknowledged	Average display time of an alarm in seconds. The time period between the alarm states "active" and "acknowledged" is calculated.

The calculation of the time of acknowledgment includes the "acknowledged" alarm state. This "acknowledged" alarm state includes the acknowledgment by the controller.

Note

For the calculation, alarms in the "acknowledged" and "inactive" states are only used if a suitable alarm with "active" state is found in the result set beforehand.

If an alarm from the controller is pending and Runtime is disabled and enabled several times, the alarm is entered into the log several times with the state "active". The alarm will also be included multiple times in the evaluation.

See also

Exporting alarms (Page 787)

6.6.6 Operating alarm statistics

Introduction

Using the statistics setup, you can change the settings for calculating the alarm statistics. The following settings are available:

Setting	Description
Time range start	• Now The current time is displayed as the start time of the calculation. • Fixed The start time of the calculation can be changed as required.
Start time	Start time for the calculation. If the "Now" option is selected under "Start time range", the start time cannot be changed.
Time range base	Unit of time for the calculation. The following settings are available: • Undefined The default time unit "Minute" is used with this setting. • Millisecond • Second • Minute • Hour • Day • Month • Year
Time range factor	The time range factor depends on the "Time range base" setting. For example, if the number 4 is set for the time range factor and "Minutes" is set for the time range base, all alarms that are logged within this period will be evaluated.

Requirement

- Alarms are located in the alarm log.
- For the following button of the alarm control, the "Visibility" and "Allow operator control" are enabled in the engineering system:

	Statistics setup

- The alarm statistics are selected in the alarm control.

Procedure

To display the statistics setup in Runtime, follow these steps:

1. Click the "Statistics setup" button in the alarm control.
   Setup opens.
2. Change the settings as required.
3. Click "OK".

Result

The calculation of the alarm statistics is displayed according to the changed settings.

6.6.7 Acknowledging alarms

Introduction

You acknowledge pending alarms in runtime depending on the configuration of your project and the defined alarm classes. The number of alarms to be acknowledged can be taken from the counter next to the "Single confirm" button or the Information bar, if this has been configured accordingly in the engineering system for the alarm control. According to the acknowledgement concept, different variants are available.

Note

If an operator authorization is configured for the operator controls, the alarms can only be acknowledged by authorized users.

Acknowledgment variants

You acknowledge individual alarms or multiple alarms together in runtime. The following options are possible:

- Single acknowledgment
  Acknowledgment of an alarm using the "Single acknowledgment" button
- Group acknowledgment
  Acknowledgment of all pending, visible alarms that require acknowledgment in the alarm control using the "Group acknowledgment" button in the alarm control.
- Acknowledgment and confirmation
  When an alarm requires acknowledgment and confirmation, you acknowledge that the alarm is incoming or outgoing. Once the alarm has gone out, you reset the alarm with the "Single confirm" button of the alarm control.

Requirement

- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Single acknowledgment
	Group acknowledgment
	Single confirm
	Show recent

Procedure

To acknowledge an alarm in runtime, follow these steps:

1. Click the "Show current" button in the alarm control.
2. Select the alarm.
3. Click "Single acknowledgment" in the alarm control.

Result

Depending on the state machine of the alarm class, the alarm receives the state "Acknowledged". The alarm can also receive the state "outgoing" if the event for triggering an alarm is no longer pending.

See also

Printing alarms in runtime (Page 793)

Configuring alarm acknowledgment (Page 749)

Displaying logged alarms in runtime (Page 779)

Group acknowledgement of alarms (Page 786)

Operating the alarm control and displaying it in runtime (Page 773)

Sorting alarms in runtime (Page 775)

Filtering alarms in runtime (Page 778)

Lock alarms (Page 791)

Shelving alarms (Page 788)

Exporting alarms (Page 787)

Alarm control (Page 309)

Acknowledgment model (Page 715)

6.6.8 Group acknowledgement of alarms

Introduction

The acknowledgement of all pending, visible alarms in the alarm window that need to be acknowledged is known as a group acknowledgement. In runtime, use the "Group acknowledgment" button in the alarm control for this purpose.

Requirement

- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Group acknowledgment

Procedure

For group acknowledgement of alarms in runtime, follow these steps:

1. Read the alarm texts of the pending alarms and perform corrective actions, if necessary.
2. Click the "Group acknowledgment" button in the alarm control.

Result

All pending alarms with the following properties have been acknowledged:

- Requires acknowledgement
- Visible

See also

Printing alarms in runtime (Page 793)

Acknowledging alarms (Page 784)

Exporting alarms (Page 787)

Operating the alarm control and displaying it in runtime (Page 773)

Sorting alarms in runtime (Page 775)

Filtering alarms in runtime (Page 778)

Displaying logged alarms in runtime (Page 779)

Shelving alarms (Page 788)

Lock alarms (Page 791)

Alarm control (Page 309)

6.6.9 Exporting alarms

Introduction

In runtime you export the data directly from the alarm control, for example, for further processing or analysis. You use the "Export" button in the alarm control for this purpose.

Requirement

- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Export

Procedure

To export data from the alarm control in runtime, follow these steps:

1. Click the "Export" button in the alarm control.
2. Under "File name" specify the file name of the export file.
3. Under "Scope of data export" specify which data is to be exported from the alarm control.
4. Under "Format" select the format of the export file.
5. Confirm with "OK".

Result

The export file appears in the browser download and can be downloaded.

See also

Group acknowledgement of alarms (Page 786)

Shelving alarms (Page 788)

Printing alarms in runtime (Page 793)

Operating the alarm control and displaying it in runtime (Page 773)

Sorting alarms in runtime (Page 775)

Filtering alarms in runtime (Page 778)

Displaying logged alarms in runtime (Page 779)

Acknowledging alarms (Page 784)

Lock alarms (Page 791)

Alarm control (Page 309)

6.6.10 Shelving alarms

Introduction

You shelve an alarm, for example, to prevent an error alarm from impairing the effectivity of your system. In runtime, use the "Shelve alarm" button in the alarm control for this purpose.

Note

If an operator authorization is configured for these control elements, the alarms can only be shelved by authorized users.

Requirement

The following setups have been made in the engineering system:

- The "Show defined alarms" alarm list is configured in such a way that shelved alarms are displayed.
- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Show defined alarms
	Shelve alarm
	Disabled alarms setup

Shelving alarms

To shelve an alarm in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Select the alarm.
3. Click the "Shelve alarm" button.

Displaying shelved alarms

To display the currently shelved alarms in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Click the "Disabled alarms setup" button.
3. Activate the option for shelved alarms.

See also

Exporting alarms (Page 787)
Lock alarms (Page 791)
Printing alarms in runtime (Page 793)
Operating the alarm control and displaying it in runtime (Page 773)
Sorting alarms in runtime (Page 775)
Filtering alarms in runtime (Page 778)
Displaying logged alarms in runtime (Page 779)
Acknowledging alarms (Page 784)
Group acknowledgement of alarms (Page 786)
Alarm control (Page 309)

6.6.11 Unshelving an alarm

Introduction

An alarm that you have shelved can be released at any time. In runtime, use the "Unshelve alarm" button in the alarm control for this purpose.

Note

If an operator authorization is configured for these control elements, the alarms can only be shelved by authorized users.

Requirement

The following setups have been made in the engineering system:

- The "Show defined alarms" alarm list is configured in such a way that shelved alarms are displayed.
  Alternatively: If the "Visibility" and "Allow operator control" settings are enabled for the "Disabled alarms setup" button, you can change the alarm list configuration in Runtime with this button.
- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Show defined alarms
	Unshelve alarm
	Disabled alarms setup

Displaying shelved alarms

To display the currently shelved alarms in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Click the "Disabled alarms setup" button.
3. Activate the option for shelved alarms.

Revoke shelved alarms

To unshelve an alarm in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Select the alarm.
3. Click "Unshelve alarm".

See also

Shelving alarms (Page 788)
Printing alarms in runtime (Page 793)
Exporting alarms (Page 787)
Lock alarms (Page 791)
Acknowledging alarms (Page 784)
Group acknowledgement of alarms (Page 786)
Displaying logged alarms in runtime (Page 779)

Sorting alarms in runtime (Page 775)

Filtering alarms in runtime (Page 778)

6.6.12 Lock alarms

Note

No locking and unlocking of PLC alarms

Locking and unlocking of PLC alarms for an S7-1500 PLC is not supported.

Introduction

You can disable alarms to avoid an excessive burden of information for the plant operator. If only selected alarms are displayed, the operator can concentrate on the important alarms.

You can enable the disabled alarms at any time. In runtime, you use the buttons "Disable alarm" and "Enable alarm" in the alarm control for this purpose.

Disabled alarms are automatically visible again when Runtime restarts.

Requirement

The following setups have been made in the engineering system:

- The "Show defined alarms" alarm list is configured in such a way that disabled alarms are displayed.
  Alternatively: If the "Visibility" and "Allow operator control" settings are enabled for the "Disabled alarms setup" button, you can change the alarm list configuration in runtime with this button.
- The user is authorized to disable and enable alarms
- The "Visibility" and "Allow operator control" settings are enabled for the following alarm control buttons:

	Show defined alarms
	Disable alarm
	Enable alarm
	Disabled alarms setup

Note

The "Disable alarms" and "Enable alarms" authorizations must be configured directly one under the other. This is necessary because the authorization level used automatically for the "Enable alarms" authorization is directly below the "Disable alarms" authorization.

Properties of disabled alarms

The following applies to disabled alarms:

- A disabled alarm is not logged.
- If a disabled alarm is reenabled, it is checked by the system and, if the cause still exists, displayed again.

Displaying disabled alarms

To display an alarm in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Click the "Disabled alarms setup" button.
3. Enable the option for disabled alarms.

Disable alarms

To disable an alarm in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Select the alarm.
3. Click the "Disable alarm" button.
   The alarm is removed from the alarm list.

Enable alarms

To enable an alarm in runtime, follow these steps:

1. Select the alarm list "Show defined alarms" in the alarm control.
2. Select the alarm.
3. Click the "Enable alarm" button.

See also

Shelving alarms (Page 788)

Printing alarms in runtime (Page 793)

Operating the alarm control and displaying it in runtime (Page 773)

Sorting alarms in runtime (Page 775)

Filtering alarms in runtime (Page 778)

Displaying logged alarms in runtime (Page 779)

Acknowledging alarms (Page 784)

Group acknowledgement of alarms (Page 786)

Exporting alarms (Page 787)

Alarm control (Page 309)

6.6.13 Printing alarms in runtime

Introduction

In runtime you print the data directly from the alarm control, for example, for further logging or analysis. You use the "Lock" button in the alarm control for this purpose.

Requirement

- The "Visibility" and "Allow operator control" settings are activated for the following alarm control buttons in the engineering system:

	Print

Procedure

To print an alarm in runtime, follow these steps:

1. Filter the alarm control using the alarm control controls, if necessary.
2. Click the "Print" button in the alarm control.
   Depending on the browser settings, the print preview appears in a new browser tab.
3. Click "Print".

Result

The alarms displayed in the alarm window are output on the printer.

See also

Acknowledging alarms (Page 784)

Operating the alarm control and displaying it in runtime (Page 773)

Group acknowledgement of alarms (Page 786)

Lock alarms (Page 791)

Shelving alarms (Page 788)
Exporting alarms (Page 787)
Displaying logged alarms in runtime (Page 779)
Filtering alarms in runtime (Page 778)
Sorting alarms in runtime (Page 775)
Alarm control (Page 309)

6.7 Display security events

6.7.1 Display security events on the HMI device

Introduction

In addition to the existing alarms in WinCC, you can also view security events on the HMI device.

Security events are, for example, an attack on a device over the network or a change of the protection level for communication between the controller and the HMI device.

Security events are detected by the controller and passed on to the HMI device. Security events are displayed in the alarm log on the HMI device.

It is not necessary to configure or activate the security event functionality within the controller. Security events are automatically detected by the controller.

Configuring the display of security events

The following steps are necessary to display security events on the HMI Device:

- Selection of controller alarms
- Creation of an alarm log for controller alarms

You can find more detailed information on configuration here: Configuring the display of security events (Page 795)

Notes

- WinCC only supports security events of a SIMATIC S7-1500 controller.
- WinCC only supports security events that are automatically updated by the central alarm management in the controller.
- Security events always use the "System information" alarm class.

See also

Configuring the display of security events (Page 795)

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Logging alarms (Page 769)

Alarm system (Page 707)

Alarms (Page 709)

6.7.2 Configuring the display of security events

Requirement

- There is an HMI connection between the HMI Device and a SIMATIC S7-1500 controller (as of firmware version 2.0).
- The option "Central alarm management in the PLC" is selected (the automatic update of security events on the HMI device is enabled in the controller).

Procedure

1. Open the "Runtime settings" of the HMI device.
2. Select the option "Automatic update" under "Alarms > Controller alarms".
   The automatic update of security events on the HMI device is enabled in the HMI device.
3. Select the option "Security events" under "Alarms > Controller alarms".
   The display by security events in runtime is enabled.

   Note

   The "Security events" option is cleared by default and must be selected for each HMI connection.

4. Create a new alarm log in the "Log" editor under "Alarm logs".
5. Open an HMI screen.
6. Create an alarm control.

Result

The security events are displayed in the alarm log in runtime.

See also

Display security events on the HMI device (Page 794)

Configuring automatic update of controller alarms on the HMI device (Page 797)

Show logged alarms (Page 765)

Displaying logged alarms in runtime (Page 779)

6.8 Sending complete alarm from the controller to the HMI device

6.8.1 Sending and automatically updating complete alarm from the controller to the HMI device

Basics

In addition to alarms in WinCC, you can configure controller alarms in STEP 7 and display them on your HMI device.

If controller alarms are configured in STEP 7, an integrated HMI connection to a SIMATIC S7-1500 controller is established and an alarm is triggered on the controller, the controller alarms are automatically sent to the HMI devices and updated automatically in case of alarm changes (e.g. change to alarm text). This will save you time because you do not have to load configuration changes of the alarms to the HMI device separately. The HMI device does not need to exit Runtime operation when the alarms are changed.

The following controller alarms can be sent to the HMI device:

- Program alarms
- ProDiag alarms
- GRAPH alarms
- System diagnostic alarms

The controller alarms can be sent completely to the HMI device if corresponding settings are configured in the controller and on the HMI device. On the HMI Device, the option "Automatic update" under "Runtime settings > Alarms > Controller alarms" must be selected for the respective connection. You can find additional information on the settings at Configuring automatic update of controller alarms on the HMI device (Page 797).

Device dependency

If the controller and the HMI device are configured accordingly, the controller alarms from the following controller are sent automatically and completely to the HMI device when they occur:

- SIMATIC S7-1500 (firmware version 2.0 and higher)

Language settings

For alarms to be displayed in the correct language on the HMI device, the same three languages or fewer must be configured for the alarms in the controller and on the HMI device. You might have to coordinate the language selection with the configuration engineer.

If different languages are configured on the HMI device and in the controller, the HMI device in operation shows the text "###Text missing###" instead of the controller alarms.

Notes

- If the "Only information" option was activated for a program alarm in STEP 7, the program alarm uses the "Information" alarm class.
- Controller alarms that are automatically updated by the central alarm management in the controller cannot be shelved or manually suppressed.
- If a security event occurs, the alarm class "system information" is used.
- The alarm number of an alarm corresponds to the alarm ID in the controller.

See also

Filtering controller alarms via display classes (Page 748)

Configuring automatic update of controller alarms on the HMI device (Page 797)

Configuring the display of system diagnostic alarms (Page 767)

Display security events on the HMI device (Page 794)

User-defined controller alarms (Page 710)

System-defined controller alarms (Page 712)

Workflow for configuring alarms (Page 724)

6.8.2 Configuring automatic update of controller alarms on the HMI device

Introduction

The "Automatic update" option is selected by default for a connection between a SIMATIC S7-1500 controller (firmware version 2.0 or higher) and an HMI device.

Requirement

- There is an HMI connection between the HMI Device and a SIMATIC S7-1500 controller (as of firmware version 2.0).
- The option "Central alarm management in the PLC" is selected in the properties of the controller (the automatic update of controller alarms on the HMI device is enabled in the controller).
- Controller alarms were configured in STEP 7.
- An alarm control is configured on the HMI device.
- The same three languages (or fewer) are configured in the controller and on the HMI device for alarms.

Procedure

1. Open the "Runtime settings" of the HMI device.
   One or more connections to controllers are displayed under "Alarms > Controller alarms".
2. Activate the "Automatic update" option for the respective connection for which you want to display the controller alarms.
   The "Automatic update" option must be selected separately for each connection.

Note

If the language of an alarm text is not available on the HMI device, the alarm text '##Text missing##' is displayed.

Result

In runtime, the controller alarms are displayed in the alarm control.

See also

Sending and automatically updating complete alarm from the controller to the HMI device (Page 796)

Filtering controller alarms via display classes (Page 748)

Configuring the display of system diagnostic alarms (Page 767)

Configuring the display of security events (Page 795)

User-defined controller alarms (Page 710)

System-defined controller alarms (Page 712)

Workflow for configuring alarms (Page 724)

6.9 Reference

6.9.1 Terminology used for alarms

WinCC	DIN 19235	DIN EN 62682	Description
Analog alarm		Absolute alarm	An alarm that is triggered with a limit value is exceeded.
Discrete alarm		Discrete alarm	An alarm that is triggered when digital signals match a specified pattern during an ongoing process.

WinCC	DIN 19235	DIN EN 62682	Description
Suppressed by design		Suppressed by design	An alarm is not signaled based on a condition.
Event		Event	An event represents a requested or unrequested status change.
Initial alarm	Initial alarm	Initial alarm	An alarm that is selected first after the last acknowledgment from several alarms.
Outgoing		Inactive	The condition for triggering an alarm is no longer fulfilled.
Outgoing alarm	Outgoing alarm	Inactive alarm	The alarm is no longer displayed because the triggering condition is no longer fulfilled.
Incoming		Active	The condition for triggering an alarm is fulfilled.
Locked alarm	Locked alarm	Locked alarm	An alarm that is blocked by interlocking.
Alarm group	Group of alarms	Alarm group	A group of alarms with a shared assignment. This assignment can be, for example, the process area or the equipment group.
Incoming alarm	Incoming alarm	Active alarm	An alarm that is displayed when the triggering condition is fulfilled.
Alarm log		Alarm log	A long-term archive for alarm logging.
Alarm filter			Criteria that limit the display of alarms in the alarm view.
Alarm list		Alarm list	A display that lists the signaled alarms with selected information (e.g. date, priority and alarm type).
Alarm type		Alarm type	An alarm attribute that allows for a distinction between the alarm condition (e.g. alarm with low or high process value).
Alarm procedures	Alarm procedures	Alarm procedures	An alarm procedure monitors the plant. The alarms of the individual alarm procedures are triggered in various ways.
Alarm	Alarm	Alarm	Acoustic or visual information about the malfunction.

WinCC	DIN 19235	DIN EN 62682	Description
Alarm view	Alarm view	Alarm view	The alarm view displays alarms that occur during the process in a plant.
Alarm class	Alarm class	Alarm class	A group of alarms with a shared set of management requirements (e.g. safety requirements).
Alarm priority	Alarm priority	Alarm priority	The relative significance that is assigned to an alarm in an alarm system that represents the urgency of the alarm.
Alarm report		Alarm report	A report of the logged or pending alarms.
New value alarm	New value alarm		An alarm processing that selects an alarm from several alarms whose status has changed since the last acknowledgment.
Acknowledgment	Acknowledgment	Acknowledgment	Operator activity that confirms the acknowledgment of the alarm.
Feedback	Feedback		The feedback confirms a command, a status or a condition. The feedback also indicates whether a change or a status change has occurred.
Deadband		Deadband	The deadband suppresses the noise component in the settled controller state of the PID controller.
Suppressed alarm	Suppressed alarm	Suppressed alarm	An alarm that is blocked by interlocking.
Alarm annunciator	Alarm annunciator	Alarm annunciator	A device or device group that alerts the operator about changes in the process conditions.

6.9.2 System events

6.9.2.1 Basics of System Events

System events

System events on the HMI device provide information about internal states of the HMI device and PLC.

The following overview illustrates when a system event occurs and how to eliminate the cause of the error.

Note

System events are output in an alarm control. System events are output in the language currently set on your HMI device.

Parameters of system events for HMI devices

System events are nested tag-based events. They are defined in a SystemData template for S7 Plus in the template for the S7 Plus HMI connection. Whenever engineering creates an S7 Plus HMI connection, a nested event is also created.

As of version V18, system events must have a constant ID of the event. Therefore, each component gets its own range of numbers.

Note

The ID of the event must be constant.

6.9.2.2 S7Plus system events

S7Plus system events

The most important system events are listed below.

ID	Name	Effect/cause	Solution
537526273	PLCDisconnectAlarm	The connection to a S7-1200/1500 PLC could not be established. Possible causes: • General connection problems (cable not plugged in, PLC switched off, network component interrupted). • Failed authentication on the PLC side (wrong or invalid password). • Connection certificate error (wrong or invalid connection certificate). • Connection resources are exhausted. 1. Error from the lower-level communication level. • Error on HMI side • Error on PLC side or network – Not connected – Access denied – Authentication failed 2. General error with the connection certificate. • OpenSSL has encountered an error while checking the connection certificate. The detail error message contains the OpenSSL error code. 3. The connection certificate used has expired. • The alarm text contains the name of the certificate. 4. The connection certificate used has been revoked. 5. The connection certificate has not been trusted, but manual trust is possible. 6. The connection certificate has not been trusted, manual trust is not possible.	Check that the S7-1200/1500 PLC and/or HMI station has no problems. Check if the cable is plugged in, the PLC is switched on and the network component is not interrupted. Check if the password is correct and valid. Check if the IP address is correct and pinging the IP address is possible. Check the S7Online access point. Check the validity of the certificate.
537526274	PLCInStopAlarm	The S7-1200/1500 PLC is not in RUN mode. The HMI station is connected to the PLC, but the internal PLC program is not executed.	Check that the S7-1200/1500 PLC has no problems. Bring the S7-1200/1500 into the RUN mode.

ID	Name	Effect/cause	Solution
537526275	FtaDisabledNotification	System diagnostic alarms have been configured for the S7-1200/1500 PLC, but the PLC does not support full-text alarms.	Check the PLC configuration. Upgrade the S7-1200/1500 PLC to a company version that supports full text messages, or deselect the "Automatic update" setting.
537526276	PlcResOverloadNotification	The S7-1200/1500 PLC communication resources for HMI tags are overloaded. The communication resources required for cyclic monitoring are more than the PLC has available. Instead, cyclic read requests are used for the resources that are not available. This leads to increased communication load. Time delays on HMI and PLC are to be expected. The alarm is triggered by an overload of 50% which is caused by the local HMI station.	Check the number of tags used simultaneously with the resources of the S7-1200/1500 PLC for cyclic jobs. Check if other HMI stations are connected to the S7-1200/1500 PLC.

6.9.2.3 Parameter set system events

2000 - System events parameter sets

The most important system events are listed below.

ID	Alarm text	Effect/causes
536920065	Parameter set [Name/ID]: Transfer to PLC successfully completed	The parameter set was successfully written from the memory to the PLC.
536920066	Parameter set [Name/ID]: Transfer to PLC aborted with error	The message appears in the following situations: • A parameter set type with the name or ID does not exist in Runtime. Transfer to the PLC has been aborted. • A parameter set with the name or ID does not exist in Runtime. Transfer to the PLC has been aborted.
536920067	Parameter set [Name/ID]: Transfer from PLC successfully completed	The parameter set was successfully loaded into memory by the PLC.
536920068	Parameter set [Name/ID]: Transfer from PLC aborted with error	The possible causes are as follows: • If a parameter set with the name or ID exists in Runtime, the values of the parameter set in the memory are overwritten with the values read by the PLC. • The transfer of PLC failed during the unsuccessful PLC connection.
536920069	Parameter set [Name/ID] not available:	A parameter set type with the name or ID is not available in Runtime.
536920070	Parameter set [Name/ID] not available	A parameter set with the name or ID is not available in memory.

ID	Alarm text	Effect/causes
536920071	Parameter set [Name/ID]: Import failed	The possible causes are as follows: • The path specified for the import file cannot be accessed. • The header in the import file is incorrect. • Specified import parameters are not available in the import file. • Checksum validation failed if checksum was activated during import.
536920072	Parameter set [Name/ID]: Import successfully completed	The parameter set was successfully imported from a file.
536920073	Parameter set [Name/ID]: Export failed	The path specified for the export file cannot be accessed.
536920080	Parameter set [Name/ID]: Export successfully completed	The parameter set was successfully exported to a file.
536920081	Invalid checksum: Import failed	Checksum validation failed due to a change in the import file.
536920082	Import successfully completed	All parameter sets were imported successfully.
536920083	Import failed	The path specified for the import file cannot be accessed. None of the parameter sets present in the import file were imported.
536920084	Import partially completed	Not all parameter sets present in the import file could be imported. A possible cause is that the values of the elements cannot be imported, e.g. the value is not within the value range of the elements or the value does not correspond to the configured data type or data format.
536920085	Export successfully completed	All parameter sets available in Runtime have been exported to the file.
536920086	Export failed	The path specified for the export file cannot be accessed.

6.9.2.4 Reporting system events

Reporting system events

The most important system events are listed below.

ID	Alarm text	Effect/causes	Solution
538640385	Initialization of the reporting service failed	Initialization of the reporting service fails.	Contact Siemens customer service.
538640386	Report Data Provider cannot be started	The data provider for reports could not be started.	Contact Siemens customer service.

ID	Alarm text	Effect/causes	Solution
538640387	The report cannot be started for the job [name].	The Report Creator for report jobs cannot be started.	Check the report job settings. If you use the "ExecuteReport" system function, check the name of the report job and the parameters passed when calling the function.
538640388	An error occurred during communication with the database server	The reporting database cannot be found or access is not possible for other reasons.	Check whether the reporting database is available at the storage location configured in the Runtime settings in engineering. Example for panel: • Is the SD card plugged in? • Does the folder specified as storage location in the Runtime settings exist? • Has the folder been specified in the correct notation?
538640389	The creation of the report job [name] failed	The Report Creator is missing information about the report job. A possible reason for this are problems with processing the report template.	Check the report job settings and the report template.
538640390	Report failed	Report Creator reports an error while generating the report.	Check the detailed error message for the report: Control "Reports" > "Reports" tab > "Status" column.

6.9.2.5 Scripting system events

Scripting system events

The most important system events are listed below.

ID	Alarm text	Effect/cause	Solution
537329665	Script debugger is activated	The script debugger for screens is enabled in the Runtime Manager.	The alarm is cleared when the script debugger for screens is disabled in the Runtime Manager.
537329666	Script debugger is activated	Script debugger for the scheduled tasks is enabled in Runtime Manager.	The alarm is fixed when the script debugger for the scheduled tasks is disabled in the Runtime Manager.
537329667	The alarm text is specified by the user via the script.	This alarm is triggered by the CreateOperatorInputInformation system function. The alarm text always comes in the language selected by the user.	--

ID	Alarm text	Effect/cause	Solution
537329668	The alarm text is specified by the user via the script.	This alarm is triggered by the CreateSystemAlarm system function. The alarm text always comes in the language selected by the user.	--
537329669	The alarm text is specified by the user via the script.	This alarm is triggered by the CreateSystemInformation system function. The alarm text always comes in the language selected by the user.	--

6.9.2.6 Communication system events

Communication system events

The most important system events are listed below.

Drivers	ID	Alarm text	Effect/cause	Solution
AllenBradleyEIP	538574849	PLCDisconnectAlarm	The connection to the Allen-Bradley PLC could not be established.	Check if: • The cable is connected • The PLC is switched on. • The network component is not interrupted.
	538574850	Computer1 [Name]: Partner is not fully operational	The PLC is not fully operational.	
Omron EIP	538574851	PLCDisconnectAlarm	The connection to the Omron PLC could not be established.	Check if: • The cable is connected • The PLC is switched on. • The network component is not interrupted.
	538574852	Computer1 [Name]: Partner is not fully operational	The PLC is not fully operational.	

Drivers	ID	Alarm text	Effect/cause	Solution
Mitsubishi MC	538574853	PLCDisconnectAlarm	The connection to the Mitsubishi MC PLC could not be established.	Check if: • The cable is connected • The PLC is switched on. • The network component is not interrupted.
	538574854	Computer1 [Name]: Partner is not fully operational	The PLC is not fully operational.	
Mitsubishi IQ	538574855	PLCDisconnectAlarm	The connection to the Mitsubishi IQ PLC could not be established.	Check if: • The cable is connected • The PLC is switched on. • The network component is not interrupted.
	538574856	Computer1 [Name]: Partner is not fully operational	The PLC is not fully operational.	
StdModbusTCP	538574857	PLCDisconnectAlarm	The connection to the Modbus PLC could not be established.	Check if: • The cable is connected • The PLC is switched on. • The network component is not interrupted.
	538574864	Computer1 [Name]: Partner is not fully operational	The PLC is not fully operational.	

6.9.2.7 VCS system events

VCS system events

The most important system events are listed below.

ID	Alarm text	Effect/cause	Solution
537264129	Computer1 (@2%s@): Manager (@1%s@) is not connected.	The alarm appears when the execution of the VCS (Visual Core Service) module is restricted. The VCS module is responsible for all processes needed to display process pictures. The possible causes are as follows: • The processes crashed. • The processes could not be started.	

6.9.2.8 Runtime system events

IOWA system events

The most important system events are listed below.

ID	Alarm text	Effect/causes	Solution
536870913	Computer1[@1%s@]: System starting	The system is started. The alarm is triggered as soon as the system starts booting and cleared when this process is completed.	The alarm is cleared automatically when the system is booted, i.e. when all managers listed in the progs file are booted (except managers that are started manually).
536870914	Computer1 [@1%s@]: System shutting down	The system is stopped. The alarm is triggered as soon as the system starts shutting down, and cleared when the system is restarted.	The alarm is cleared automatically when the system starts up again.
536870915	Computer1[@S1%s@]: Delta activation in progress	The alarm indicates that a delta download is in progress. The alarm is set when the delta download is started and remains active until the delta activation is completed.	The alarm disappears automatically as soon as the delta download is completed. No action is required by the operator.
536870916	Computer1 [@S1%s@]: Delta rollback in progress	The alarm appears when the delta activation has failed and a rollback of the changes has been triggered. It remains active while the rollback is executed and is reset afterwards.	The alarm disappears automatically as soon as the rollback and therefore the failed delta download is completed. No action is required by the operator.
536870917	Computer1 [@1%s@]: Low work memory	The alarm indicates that the free physical work memory (RAM) will soon be exhausted.	Increase the work memory (RAM) or reduce the load.
536870918	Computer1 [@1%s@]: Very low work memory	The alarm indicates that the free physical work memory (RAM) is critical.	Increase the work memory (RAM) or reduce the load.

ID	Alarm text	Effect/causes	Solution
536870919	Computer1 [@1%s@]: Low hard disk space	The alarm indicates that the free memory space will soon be exhausted.	Delete the data that is not required.
536870920	Computer1 [@1%s@]: Very low hard disk space	The alarm indicates that the free memory space is no longer sufficient.	Delete the data that is not required.
536870921	Computer1 [@2%s@]: Manager [@1%s@] is not connected.	The alarm indicates that the connection to the Service Manager has been lost.	Check the network connection or the status of the manager. The system attempts to establish a connection on its own.
536870922	Computer1 [@2%s@]: Service [@1%s@] is not being executed.	The alarm indicates that the connection to a service instance has been lost.	Check the network connection or the status of the manager. The system attempts to establish a connection on its own.
536870923	Online backup in progress	The alarm is displayed when the online backup operation is performed. It remains set until the online backup is completed.	The alarm disappears automatically when the online backup is completed or aborted with an error. No action is required by the operator.
536870924	SystemManagerAlarm	The alarm indicates that the System Manager has its lost connection to the Event Manager.	Check the network connection or the status of the manager. The system attempts to establish a connection on its own.
536870925	RedundancyLossAlarm	The alarm indicates that there is currently no established connection to the redundant partner. This may be because the connection to the other host has been lost or because the connection has not yet been established (other host not running, network problem).	Check the network connection and whether the other host was started with the correct configuration. The system attempts to establish a connection on its own.
536870926	RuntimeVersionDifferent	The alarm appears when a redundantly configured system is started with two different Runtime versions on the two hosts. The "RuntimeVersion" configuration entry is different on the peers, which means that different software versions are installed on the two hosts.	Check the Runtime version value in the configurations on both devices and ensure that the correct software versions are being used. Update the incorrect hosts to the correct Runtime version.
536870927	Computer1 [@S1%s@]: Connection ID: @S7%u@, Connection status: @2%u@	The alarm indicates that the status of the connection between the driver and the PLC has changed.	
536870928	Computer1 [@S1%s@]: @5%t#4T@	The alarm appears when a configuration error has been detected by a component.	The component should describe the problem in detail so that the user can make the required changes in the configuration.

ID	Alarm text	Effect/causes	Solution
536870929	Computer1 [@S1%s@]: Successful rollback after failed activation.	The alarm is displayed when a delta download attempt fails and the changes are not activated.	There can be several reasons why the delta download process fails. The detailed reason should help identify the problem. Common problems include configuration issues within the delta project, incompatibility with the currently running project, requirements for a delta download are not met (such as a mandatory manager not running), another process is running that is preventing the delta download from running.
536870930	Computer1 [@S1%s@]: Delta rollback failed	The alarm is displayed when a problem is detected in a component during a delta download and a rollback is initiated. However, this rollback also fails for unexpected reasons. Consequently, this may result in a project that is in an inconsistent state.	The project may be in an inconsistent state. Perform a full download to return to a consistent project.
536870931	Computer1 [@S1%s@]: Saving of configuration changes has failed.	The alarm is displayed when saving configuration changes fails. Writing the changes was unsuccessful and the changes are lost after shutdown.	Check whether the required write rights have been granted for the project folder.
536870932	Computer1 [@S1%s@]: Merging of the RDF files has failed: @2%s@	The alarm indicates that the Runtime configuration data consolidation (RDF merge) failed.	Verify that the required authorizations have been granted for the project folder and that the files in the project are not locked by other processes.
536870933	Computer1 [@S1%s@]: Delta activation has failed. The file system is damaged.	The alarm is activated when a delta download was successfully activated, but the delta build number could not be added to the permanent list of activated deltas.	Verify that the required authorizations have been assigned to the project folder and that the files in the project are not locked by other processes, especially the delta list file in the deltas subdirectory.
536870934	Computer1 [@S1%s@]: A leap in time of @2%1.3f@ seconds was determined.	The alarm indicates that a time jump has occurred. Do not change the system time.	Check the device battery and system time.

ID	Alarm text	Effect/causes	Solution
536870935	Runtime Collaboration: @2%t#<RuntimeCollaborationFailureReason.1>@ during the connection of computer [@S1%s@] with system @5%u@ [@6%s@] to @4%s@.	The alarm indicates an unsuccessful connection attempt of the Dist Manager to another system with the cause of the error specified in the alarm text. There can be several reasons why a connection attempt fails. Common problems that can occur: Target system not available, certificate problems, wrong system ID or name configured for the target, time synchronization problem.	The alarm text, which includes the reason for the error and information about the connection attempt, should help identify the problem. - If the target system is not available: Check if the target system is running with correct configurations (Dist and, if necessary, Proxy Manager are started), check for network errors (check if the device itself is reachable, the ports used are open) - Certificate problems: Check if both client and server system have correct and valid certificates. Incorrect system ID or name: The client system has a mismatched configuration to the target system (server), where either the system ID or name differs from the actual ID or name of the system. - Time synchronization problem: The system time difference between the two systems is greater than the maximum allowed difference.
536870936	Computer1 [@2%s@]: Loading of the configuration data created in Runtime has failed: @3%s@	The alarm indicates that configuration data created through Runtime engineering or by the Runtime itself could not be loaded. This data is stored in Runtime configuration files within the Runtime project and is read when Runtime is restarted.	Usually indicates a serious problem. Problems with the storage medium itself or with corrupted files.
536870937	Computer1 [@S1%s@]: 'LoggingOverloadError' event was activated.	Outdated	Instead, use StorageSystemWriteDataLostAlarm_[StorageApplicationAbbreviation].
536870938	Computer1 [@S1%s@]: User name: @S2%s@, Tag name: @2%s@, Old value: @3%v@, New value: @4%v@, Unit: @5%s@, Cause: @6%s@	A value was changed by the user.	Audit Trail value change documentation.

ID	Alarm text	Effect/causes	Solution
536870939	Computer1 [@S1%s@]: User name: @S2%s@, Tag name: @2%s@, Old value: @3%v@, New value: @4%v@, Unit: @5%s@, Cause: @6%s@	A value change by the user has not occurred.	Audit Trail value change documentation.
536870940	Computer1 [@S1%s@]: Online backup has failed	The alarm is triggered when an online backup was requested, but the operation fails.	The backup may fail for several reasons, which can be found in the detailed description. A mandatory manager may not be running, or may have been stopped during the backup, or there may be an access rights problem in the target directory. Certain components may have also implemented their own logic for online backup that fails.
536870941	Computer1 [@S1%s@]: The certificate for @2%s@ was not found in the certificate memory. Details: @3%s@	The alarm is triggered when a component cannot use certificates due to problems. Either no certificate is configured or there is a problem with the public and/or private keys.	Check the configured certificates used by this component.
536870942	Computer1 [@S1%s@]: The certificate for @2%s@ will expire soon. Expiration date: @3%s@. Details: @4%s@	The alarm is triggered when a component is using a certificate that is about to expire. The exact time depends on what the specific component has defined as soon to expire. Usually, this alarm is expected to be triggered about 1 week before the expiration date.	Update the certificates used by the respective component.
536870943	Computer1 [@S1%s@]: The certificate for @2%s@ has expired. Expiration date: @3%s@. Details: @4%s@	The alarm is triggered when a component attempts to use an expired certificate.	Update the certificates used by the respective component.
536870944	SystemDistributionManagerAlarm	The alarm indicates that the connection to the Distribution Manager has been lost.	Check the network connection or the status of the manager. The system attempts to establish a connection on its own.

ID	Alarm text	Effect/causes	Solution
536870945	Computer1 [@2%s@]: Driver [@1%s@] Overload warning.	The alarm indicates that the driver can no longer continuously process the number of value changes it receives from the PLC. This state occurs either only temporarily, or can also exist permanently. In this case, received values are temporarily stored by the driver and processed at a later time.	If this occurs only temporarily, it is enough to wait. If this occurs more frequently, then using another driver can share the load between the two drivers and thus prevent the problem. It could also be an indication that the event manager can no longer handle the number of value changes. In this case, the driver is storing the value changes, which in turn can cause an overload condition with this driver.
536870946	Computer1 [@2%s@]: Driver [@1%s@} Overload has occurred.	The alarm indicates that the driver can no longer handle the number of value changes it receives from the PLC. This condition either occurs only temporarily, or may continue permanently. In this case, it cannot be excluded that values are lost.	If this occurs only temporarily, it is enough to wait. If this occurs more frequently, then using another driver can share the load between the two drivers and thus prevent the problem. It could also be an indication that the event manager can no longer handle the number of value changes. In this case, the driver is storing the value changes, which in turn can cause an overload condition with this driver.
536870948	Computer1 [@2%s@]: Manager [@1%s@] is not connected.	The alarm indicates that the driver has lost the connection to the Event Manager.	Check the network connection or the status of the manager. The system attempts to establish a connection on its own.
536870949	DiskSpaceWarning	The alarm indicates that the configured limit for free space on the external storage medium has been exceeded.	Delete the data that is not required. Expand the external storage medium.
536870950	DiskSpaceAlarm	The alarm indicates that the free space on the external storage medium is no longer sufficient.	Delete the data that is not required. Expand the external storage medium.
536870951	RemovableStorageMediumAlarm	The alarm indicates that a removable device has been removed.	The alarm can be disabled for permanently installed data storage media.
536870952	High limit violated for tag @1%s@. Computer name: @S1%s@	High limit violation for a tag value range.	The tag value is above the defined value range - check the cause.
536870953	Low limit violated for tag @1%s@. Computer name: @S1%s@	Low limit violation of a tag value range.	The tag value is below the defined value range - check the cause.
536870954	MissingCRLAlarm	The alarm is triggered when the component that monitors CRLs cannot find the CRLs for CA certificate file(s). The subject names of all relevant CAs are specified in the alarm.	Specify the missing CRLs.
536870955	ExpiredCRLAlarm	The alarm is triggered by the component that monitors CRLs when expired CRLs have been provided. The subject names of all relevant CAs are specified in the alarm.	Renew the CRLs used.

ID	Alarm text	Effect/causes	Solution
536871425	Computer1 [@5%s@]: @3%t#2T@	The connection to the S7 PLC was interrupted.	Check the S7 PLC and/or driver for problems. Check if the IP address of the PLC is correct and pinging the IP address is possible.
536871426	Computer1 [@2%s@]: Partner is not fully operational.	The S7 PLC is in Stop mode. The driver is connected to the PLC, but the internal PLC program is not executed.	Check the S7 PLC for problems and bring the S7 to the "Run" mode if necessary.
536871681	PlcDisconnectedAlarm_OPCUA	The connection to the OPC UA server was interrupted.	Check the OPC UA server and/or driver for problems. Check the connection between the OPC UA server and/or driver, especially if it is not blocked by a firewall.
536871937	Computer1 [@3%s@]: @1%s@ (@2%s@): has no connection to the logging system.	AlarmLogging service has lost the connection to the database. This is probably due to an internal error of the database. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the database server. In some cases, starting/restarting the database may solve the problem. Otherwise, the database log files can help to find the cause.
536871938	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Log is full.	At least one storage medium used by the AlarmLogging service is full, so new value changes cannot be saved. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the connected storage media and free some space on the devices that are (almost) full. Changing the log configuration can help reduce the amount of disk space needed by limiting the maximum amount of space a particular log can use before deleting data.
536871939	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Insufficient resources of the logging system.	The database used by the AlarmLogging service is still running, but has reported that it has run out of resources (e.g. work memory, disk space). No more writing is possible. (Only applicable with Process Historian.)	Check the storage medium if it is out of space. The database server logs can also help to find out the cause of the problem. Storage space problems can be caused by a high load that the system cannot handle.
536871940	Computer1 [@3%s@]: @1%s@: A write buffer overflow has occurred in log @2%s@. Data will be lost.	In the AlarmLogging service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. With redundant deployment, if only one host has this problem temporarily, the data from the other host is synchronized once the problem is resolved.	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project.

ID	Alarm text	Effect/causes	Solution
536871941	Computer1 [@2%s@]: @1%s@: The passive logging system has lost the connection to the active logging system.	The redundant databases used by the AlarmLogging service have lost connection to each other (triggered only on the passive host). Possible causes: • The active database is completely shut down. • A problem with the network connection. If the StorageSystemPassiveSynchronizing alarm is also triggered, it means that the problem is corrected and the alarm is cleared as soon as synchronization is completed. (Only applicable with Process Historian.)	Check the status of the active database. If it does not run, try restarting it or analyze its log files to learn more about the problem. If the database is running, ensure that the network connection between the two hosts is working.
536871942	Computer1 [@2%s@]: @1%s@: Synchronizing data	Logged data synchronization is in progress between the active and passive databases. Redundancy failover during this state may result in loss of logged data. (Only applicable with Process Historian.)	This system event does not signify an error; it is normal behavior when the two redundant databases are synchronizing data. Therefore, no action is required. As long as this system event is pending, you should not initiate redundancy failover, otherwise the logged data may be lost.
536871943	Computer1 [@2%s@]: @1%s@: Failover error	AlarmLogging service: An error occurred while the database was trying to switch roles between the active and passive databases. (Only applicable with Process Historian.)	Analyze the Process Historian log files to learn more about the reason for the failure. If other system events are also triggered, they may also be related.
536871944	Computer1 (@3%s@): @1%s@ (@2%s@) No connection to the logging system backend	AlarmLogging service: The backend of the database was shut down. It is either completely down or the corresponding network connection has problems. (Only applicable with Process Historian.)	Check the status of the database backend. If it does not work, try restarting it or analyze the log files to learn more about the problem. Otherwise, ensure that the network connection is working.
536871945	Computer1 [@3%s@] - @1%s@: An overflow of the transaction buffer and write buffer has occurred in log @2%s@ of the passive partner. Data may be lost. Avoid a manual failover of the redundancy partner while this alarm is active.	In the passive AlarmLogging service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. If the connection between the redundant services is interrupted, this can also cause this problem. If the active service has no problems, this alarm does not mean real data loss. (Only applicable with Process Historian.)	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project. If the connection to the active service has been lost, but it is working without any problems, it is important not to initiate redundancy failover after the connection is restored while this alarm is still present to avoid real data loss.

ID	Alarm text	Effect/causes	Solution
536871946	Computer1 [@3%s@] - @1%s@: (@2%s@) Connection to the long-term logging system is lost.	AlarmLogging service has its lost connection to the long-term storage system. This is probably due to an internal database error or a network problem. (Only applicable with Process Historian.)	Check the status of the long-term database server. Starting/restarting the server may solve the problem. If it is running, ensure that the network connection is working. Otherwise, the database logs may help find the cause.
536871947	Computer1 [@4%s@] - @1%s@ (@2%s@ - @3%s@): Long-term logging system is out of hard disk space.	At least one storage medium used by the AlarmLogging service long-term storage system is full, so new value changes cannot be saved. (Only applicable with Process Historian.)	Check the status of the connected storage media on the long-term storage system and free up some space on the devices (almost) full. Changing the log configuration can help reduce the amount of disk space needed by limiting the maximum amount of space a particular log can use before deleting the data.
536871948	Computer1 [@3%s@] - @1%s@: (@2%s@): Long-term logging system no longer has any system resources.	The long-term storage system used by the AlarmLogging service is still running, but has reported that it is out of resources (e.g. memory, disk space). No more writing is possible. (Only applicable with Process Historian.)	Check the long-term storage media to see if they are running out of space. The database server logs can also help to find out the cause of the problem. Storage space problems can be caused by a high load that the system cannot handle.
536871949	Computer1 [@2%s@] - @1%s@: The passive and the active long-term logging system are no longer synchronized.	The redundant long-term databases used by AlarmLogging service have lost connection to each other (triggered only on the passive host). Possible causes: • The active database is completely shut down. • A problem with the network connection. (Only applicable with Process Historian.)	Check the status of the active long-term storage system. If it is down, try to restart it or analyze its logged files to learn more about the problem. If the database is running, ensure that the network connection between the two hosts is working.
536871950	Computer1 [@2%s@] - @1%s@: Synchronizing long-term logging system	AlarmLogging service: Synchronization of logged data between the active and passive long-term storage systems is in progress. Redundancy failover during this state may result in loss of logged data. (Only applicable with Process Historian.)	This system event does not signify an error; it is normal behavior when the two redundant databases are synchronizing data. Therefore, no action is required. As long as this system event is pending, you should not initiate redundancy failover, otherwise the logged data may be lost.
536871951	Computer1 [@2%s@] - @1%s@: Failover error in the long-term logging system.	AlarmLogging service: An error occurred when the long-term storage system tried to switch roles between the active and passive databases. (Only applicable with Process Historian.)	Analyze the Process Historian log files to learn more information about the reason for the failure.

ID	Alarm text	Effect/causes	Solution
536871952	Computer1 [@3%s@] - @1%s@: (@2%s@) A general error has occurred. Check the traces for additional information.	AlarmLogging service: An unknown error occurred during the execution of a database operation.	Analyze previous traces to learn more about the problem. One possible scenario that can cause this problem is the unexpected removal of a storage medium. In this case, reinstall the storage medium and restart Runtime. Otherwise, contact Siemens support for assistance.
536871953	Computer1 [@6%s@] - @1%s@ (@2%s@ - @3%s@): Logging medium cannot be reached. @5%t#4T@	AlarmLogging service: When trying to create a database or reconnect to a storage medium, a storage medium was not available or authorizations were lacking.	Check if the configuration (path of the log) is valid for all logs and if all required storage media are connected and working. If authorizations are available, ensure they are set correctly for the file system.
536871954	Computer1 [@2%s@]: The logging system @1%s@ has detected a more recent database version that is not compatible with the current runtime version. Update the runtime version on your device or use a device with the same runtime version as the database version.	AlarmLogging service: The Runtime software used is older than the databases used. Probably the database was moved from a device with newer software to a device with older software.	Move the database to a device with the same or a newer version of the Runtime software, or upgrade the software version of the device.
536871955	Computer1 [@2%s@]: @1%s@: Synchronizing the logging data of the redundant partners.	The redundant AlarmLogging services synchronize data with each other. This is normal behavior that only serves as an indication that the historical data on both hosts is not quite up-to-date.	No action is required, but as long as this system event persists, it is recommended not to shut down any of the hosts. If this alarm has persisted since the last start-up/restart of one of the hosts, do not restart or shut down the longer running host or data loss may occur.

ID	Alarm text	Effect/causes	Solution
536871956	Computer1 [@2%s@]: @1%s@: Error during background synchronization. Check the traces for additional information.	AlarmLogging service: An error occurred during data synchronization.	Check the additional information of the alarm to identify and eliminate the cause of the error. The synchronization is repeated at regular intervals so that it can be performed shortly after the cause has been eliminated.
536872193	Computer1 [@3%s@]: @1%s@ (@2%s@): has no connection to the logging system.	AlarmPersistency service has its lost connection to the database. Possible cause: Internal error of the database. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the database. Otherwise, the log files may help find the cause.
536872194	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Log is full.	At least one storage medium used by the AlarmPersistency service is full, so new value changes cannot be saved. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the connected storage media and free up some space on the devices (almost) full.
536872195	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Insufficient resources of the logging system.	Not applicable.	
536872196	Computer1 [@3%s@]: @1%s@: A write buffer overflow has occurred in log @2%s@. Data will be lost.	In the AlarmPersistency service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. With redundant deployment, if only one host has this problem temporarily, the data from the other host is synchronized once the problem is resolved.	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system alarm. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project.
536872197	Computer1 [@2%s@]: @1%s@: The passive logging system has lost the connection to the active logging system.	Not applicable.	

ID	Alarm text	Effect/causes	Solution
536872198	Computer1 [@2%s@]: @1%s@: Synchronizing data	Not applicable.	
536872199	Computer1 [@2%s@]: @1%s@: Failover error	Not applicable.	
536872200	Computer1 (@3%s@): @1%s@ (@2%s@) No connection to the logging system backend	Not applicable.	
536872201	Computer1 [@3%s@] - @1%s@: An overflow of the transaction buffer and write buffer has occurred in log @2%s@ of the passive partner. Data may be lost. Avoid a manual failover of the redundancy partner while this alarm is active.	Not applicable.	
536872202	Computer1 [@3%s@] - @1%s@: (@2%s@) Connection to the long-term logging system is lost.	Not applicable.	
536872203	Computer1 [@4%s@] - @1%s@ (@2%s@ - @3%s@): Long-term logging system is out of hard disk space.	Not applicable.	

ID	Alarm text	Effect/causes	Solution
536872204	Computer1 [@3%s@] - @1%s@: (@2%s@): Long-term logging system no longer has any system resources.	Not applicable.	
536872205	Computer1 [@2%s@] - @1%s@: The passive and the active long-term logging system are no longer synchronized.	Not applicable.	
536872206	Computer1 [@2%s@] - @1%s@: Synchronizing long-term logging system	Not applicable.	
536872207	Computer1 [@2%s@] - @1%s@: Failover error in the long-term logging system.	Not applicable.	
536872208	Computer1 [@3%s@] - @1%s@: (@2%s@) A general error has occurred. Check the traces for additional information.	AlarmPersistency service: An unknown error occurred during the execution of a database operation.	Analyze previous traces to learn more about the problem. One possible scenario that can cause this problem is the unexpected removal of a storage medium. In this case, reinstall the storage medium and restart Runtime. Otherwise, contact Siemens support for assistance.
536872209	Computer1 [@6%s@] - @1%s@ (@2%s@ - @3%s@): Logging medium cannot be reached. @5%t#4T@	AlarmPersistency service: When trying to create a database or reconnect to a storage medium, no storage medium was available or authorizations were lacking.	Check if the configuration (path of the AlarmPersistency database) is valid and if all required storage media are connected and working. If available, ensure that the file system authorizations are set correctly.

ID	Alarm text	Effect/causes	Solution
536872210	Computer1 [@2%s@]: The logging system @1%s@ has detected a more recent database version that is not compatible with the current runtime version. Update the runtime version on your device or use a device with the same runtime version as the database version.	AlarmPersistency service: The Runtime software used is older than the databases used. Probably the database was moved from a device with newer software to a device with older software.	Move the database to a device with the same or a newer version of the Runtime software or update the software version of the device.
536872211	Computer1 [@2%s@]: @1%s@: Synchronizing the logging data of the redundant partners.	Not applicable.	
536872212	Computer1 [@2%s@]: @1%s@: Error during background synchronization. Check the traces for additional information.	Not applicable.	
536872449	Computer1 [@3%s@]: @1%s@ (@2%s@): has no connection to the logging system.	TagLogging service has lost its connection to the database. Possible cause: Internal error of the database. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the database server. In some cases, starting/restarting the database may solve the problem. Otherwise, the database log files can help to find the cause.
536872450	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Log is full.	At least one storage medium used by the TagLogging service is full, so new value changes cannot be saved. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the connected storage media and free some space on the devices (almost) full. Changing the log configuration can help reduce the amount of disk space needed by limiting the maximum amount of space a particular log can use before deleting data.

ID	Alarm text	Effect/causes	Solution
536872451	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Insufficient resources of the logging system.	The database used by the TagLogging service is still running, but has reported that it is out of resources (e.g. work memory, disk space). No more writing is possible. (Only applicable with Process Historian.)	Check the storage medium if it is out of space. The database server logs can also help to find out the cause of the problem. Storage space problems can be caused by a high load that the system cannot handle.
536872452	Computer1 [@3%s@]: @1%s@: A write buffer overflow has occurred in log @2%s@. Data will be lost.	In the TagLogging service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. With redundant deployment, if only one host has this problem temporarily, the data from the other host is synchronized once the problem is resolved.	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project.
536872453	Computer1 [@2%s@]: @1%s@: The passive logging system has lost the connection to the active logging system.	The redundant databases used by the TagLogging service have lost connection to each other (triggered only on the passive host). Possible causes: • The active database is completely shut down. • A problem with the network connection. If the StorageSystemPassiveSynchronizing alarm is also triggered, it means that the problem is corrected and the alarm will be cleared as soon as synchronization is completed. (Only applicable with Process Historian.)	Check the status of the active database. If the database is not running, try restarting it or analyze its log files to learn more about the problem. If the database is running, ensure that the network connection between the two hosts is working.
536872454	Computer1 [@2%s@]: @1%s@: Synchronizing data	TagLogging service: Logged data synchronization is in progress between the active and passive databases. Redundancy failover during this state may result in loss of logged data. (Only applicable with Process Historian.)	This system event does not signify an error; it is normal behavior when the two redundant databases are synchronizing data. Therefore, no action is required. As long as this system event is pending, you should not initiate redundancy failover, otherwise the logged data may be lost.
536872455	Computer1 [@2%s@]: @1%s@: Failover error	TagLogging service: An error occurred while the database was trying to switch roles between the active and passive databases. (Only applicable with Process Historian.)	Analyze the Process Historian log files to learn more about the reason for the failure. If other system events are also triggered, they may also be related.
536872456	Computer1 (@3%s@): @1%s@ (@2%s@) No connection to the logging system backend	TagLogging service: The backend of the database was shut down. It is either completely down or the network connection to it is having problems. (Only applicable with Process Historian.)	Check the status of the database backend. If it does not work, restart it or analyze the log files to learn more about the causes. Ensure that the network connection is working.

ID	Alarm text	Effect/causes	Solution
536872457	Computer1 [@3%s@] - @1%s@: An overflow of the transaction buffer and write buffer has occurred in log @2%s@ of the passive partner. Data may be lost. Avoid a manual failover of the redundancy partner while this alarm is active.	In the passive TagLogging service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. • Interrupted connection between the redundant services. If the active service has no problems, this alarm does not mean real data loss. (Only applicable with Process Historian.)	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project. If the connection to the active service has been lost, but it is working without any problems, it is important not to initiate redundancy failover after the connection is restored while this alarm is still present to avoid real data loss.
536872458	Computer1 [@3%s@] - @1%s@: (@2%s@) Connection to the long-term logging system is lost.	TagLogging service has its lost connection to the long-term storage system. Possible causes: • Internal database error. • Network problem. (Only applicable with Process Historian.)	Check the status of the long-term database server. Start/restart the server. If it is running, ensure that the network connection is working. Database logs can also help to find the cause.
536872459	Computer1 [@4%s@] - @1%s@ (@2%s@ - @3%s@): Long-term logging system is out of hard disk space.	At least one storage medium used by the TagLogging service long-term storage system is full, so new value changes cannot be saved. (Only applicable with Process Historian.)	Check the status of the connected storage media on the long-term storage system and free up some space on the devices (almost) full. Changing the log configuration can help reduce the amount of disk space needed by limiting the maximum amount of space a particular log can use before deleting the data.
536872460	Computer1 [@3%s@] - @1%s@: (@2%s@): Long-term logging system no longer has any system resources.	The long-term storage system used by the TagLogging service is still running, but has reported that it is out of resources (e.g. memory, disk space). No more writing is possible. (Only applicable with Process Historian.)	Check the long-term storage media to see if they are running out of space. The database server logs can also help to find out the cause of the problem. Storage space problems can be caused by a large load that the system cannot handle.
536872461	Computer1 [@2%s@] - @1%s@: The passive and the active long-term logging system are no longer synchronized.	The redundant long-term databases used by the TagLogging service have lost connection to each other (triggered only on the passive host). Possible causes: • The active database is completely shut down. • A problem with the network connection. (Only applicable with Process Historian.)	Check the status of the active long-term storage system. If it is down, try to restart it or analyze its logged files to learn more about the problem. If the database is running, ensure that the network connection between the two hosts is working.

ID	Alarm text	Effect/causes	Solution
536872462	Computer1 [@2%s@] - @1%s@: Synchronizing long-term logging system	TagLogging service: Synchronization of logged data between the active and passive long-term storage systems is in progress. Redundancy failover during this state may result in loss of logged data. (Only applicable with Process Historian.)	This system event does not signify an error; it is normal behavior when the two redundant databases are synchronizing data. Therefore, no action is required. As long as this system event is pending, you should not initiate redundancy failover, otherwise the logged data may be lost.
536872463	Computer1 [@2%s@] - @1%s@: Failover error in the long-term logging system.	TagLogging service: An error occurred while the long-term storage system was trying to switch roles between the active and passive databases. (Only applicable with Process Historian.)	Analyze the Process Historian log files to learn more about the reason for the failure.
536872464	Computer1 [@3%s@] - @1%s@: (@2%s@) A general error has occurred. Check the traces for additional information.	TagLogging service: An unknown error occurred during the execution of a database operation.	Analyze previous traces to learn more about the problem. One possible cause of this problem is the unexpected removal of a storage device. In this case, reinstall the storage medium and restart Runtime. Otherwise, contact Siemens support for assistance.
536872465	Computer1 [@6%s@] - @1%s@ (@2%s@ - @3%s@): Logging medium cannot be reached. @5%t#4T@	TagLogging service: When trying to create a database or reconnect to a storage medium, a storage medium was not available or authorizations were lacking.	Check if the configuration (path of the log) is valid for all logs and if all required storage media are connected and working. If authorizations are available, ensure that the file system authorizations are set correctly.
536872466	Computer1 [@2%s@]: The logging system @1%s@ has detected a more recent database version that is not compatible with the current runtime version. Update the runtime version on your device or use a device with the same runtime version as the database version.	TagLogging service: The Runtime software used is older than the databases used. Probably the database was moved from a device with newer software to a device with older software.	Move the database to a device with the same or a newer version of the Runtime software or update the software version of the device.

ID	Alarm text	Effect/causes	Solution
536872467	Computer1 [@2%s@]: @1%s@: Synchronizing the logging data of the redundant partners.	The redundant TagLogging services synchronize the data with each other. This is normal behavior that only indicates that the historical data on both hosts is not quite up to date.	No action is required, but as long as this system event persists, it is recommended not to shut down any of the hosts. If this alarm has persisted since the last start-up/restart of one of the hosts, do not restart or shut down the longer running host or data loss may occur.
536872468	Computer1 [@2%s@]: @1%s@: Error during background synchronization. Check the traces for additional information.	TagLogging service: An error occurred during data synchronization.	Check the additional information of the alarm to identify and eliminate the cause of the error. The synchronization is repeated at regular intervals so that it can be performed shortly after the cause has been eliminated.
536872705	Computer1 [@3%s@]: @1%s@ (@2%s@): has no connection to the logging system.	TagPersistency service has its lost connection to the database. Possible cause: • Internal database error. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the database. The log files of the Runtime system can also help to find the cause.
536872706	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Log is full.	At least one storage medium used by the TagPersistency service is full, so new value changes cannot be saved. If this situation continues for too long, the buffers for values can become full, resulting in data loss.	Check the status of the connected storage media and free up some space on the devices (almost) full.
536872707	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Insufficient resources of the logging system.	Not applicable.	
536872708	Computer1 [@3%s@]: @1%s@: A write buffer overflow has occurred in log @2%s@. Data will be lost.	In the TagPersistency service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. With redundant deployment, if only one host has this problem temporarily, the data from the other host is synchronized once the problem is resolved.	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project.

ID	Alarm text	Effect/causes	Solution
536872709	Computer1 [@2%s@]: @1%s@: The passive logging system has lost the connection to the active logging system.	Not applicable.	
536872710	Computer1 [@2%s@]: @1%s@: Synchronizing data	Not applicable.	
536872711	Computer1 [@2%s@]: @1%s@: Failover error	Not applicable.	
536872712	Computer1 (@3%s@): @1%s@ (@2%s@) No connection to the logging system backend	Not applicable.	
536872713	Computer1 [@3%s@] - @1%s@: An overflow of the transaction buffer and write buffer has occurred in log @2%s@ of the passive partner. Data may be lost. Avoid a manual failover of the redundancy partner while this alarm is active.	Not applicable.	
536872714	Computer1 [@3%s@] - @1%s@: (@2%s@) Connection to the long-term logging system is lost.	Not applicable.	

ID	Alarm text	Effect/causes	Solution
536872715	Computer1 [@4%s@] - @1%s@ (@2%s@ - @3%s@): Long-term logging system is out of hard disk space.	Not applicable.	
536872716	Computer1 [@3%s@] - @1%s@: (@2%s@): Long-term logging system no longer has any system resources.	Not applicable.	
536872717	Computer1 [@2%s@] - @1%s@: The passive and the active long-term logging system are no longer synchronized.	Not applicable.	
536872718	Computer1 [@2%s@] - @1%s@: Synchronizing long-term logging system	Not applicable.	
536872719	Computer1 [@2%s@] - @1%s@: Failover error in the long-term logging system.	Not applicable.	
536872720	Computer1 [@3%s@] - @1%s@: (@2%s@) A general error has occurred. Check the traces for additional information.	TagPersistency service: An unknown error occurred during the execution of a database operation.	Analyze previous traces to learn more about the problem. One possible cause is the unexpected removal of a storage medium. In this case, reinstall the storage medium and restart Runtime. Otherwise, contact Siemens support for assistance.

ID	Alarm text	Effect/causes	Solution
536872721	Computer1 [@6%s@] - @1%s@ (@2%s@ - @3%s@): Logging medium cannot be reached. @5%t#4T@	TagPersistency service: When trying to create a database or reconnect to the storage medium, no storage medium was available or authorizations were lacking.	Check if the configuration (path of the TagPersistency database) is valid and if all required storage media are connected and working. If authorizations are available, ensure that the file system authorizations are set correctly.
536872722	Computer1 [@2%s@]: The logging system @1%s@ has detected a more recent database version that is not compatible with the current runtime version. Update the runtime version on your device or use a device with the same runtime version as the database version.	TagPersistency service: The Runtime software used is older than the databases used. Probably the database was moved from a device with newer software to a device with older software.	Move the database to a device with the same or a newer version of the Runtime software or update the software version of the device.
536872723	Computer1 [@2%s@]: @1%s@: Synchronizing the logging data of the redundant partners.	Not applicable.	
536872724	Computer1 [@2%s@]: @1%s@: Error during background synchronization. Check the traces for additional information.	Not applicable.	
536872961	Computer1 [@3%s@]: @1%s@ (@2%s@): has no connection to the logging system.	ContextLogging service has its lost connection to the database. Possible cause: • Internal database error. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the database server. Start/restart database. The log files of the database can also help to find the cause.

ID	Alarm text	Effect/causes	Solution
536872962	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Log is full.	At least one storage medium used by Context-Logging service is full, so new value changes cannot be saved. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the connected storage media and free up some space on the devices (almost) full. Changing the log configuration can help reduce the amount of disk space needed by limiting the maximum amount of space a particular log can use before deleting the data.
536872963	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Insufficient resources of the logging system.	Not applicable.	
536872964	Computer1 [@3%s@]: @1%s@: A write buffer overflow has occurred in log @2%s@. Data will be lost.	In the ContextLogging service, the buffers are full and therefore some data has already been lost. Possible causes: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. With redundant deployment, if only one host has this problem temporarily, the data from the other host is synchronized once the problem is resolved.	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project.
536872965	Computer1 [@2%s@]: @1%s@: The passive logging system has lost the connection to the active logging system.	Not applicable.	
536872966	Computer1 [@2%s@]: @1%s@: Synchronizing data	Not applicable.	
536872967	Computer1 [@2%s@]: @1%s@: Failover error	Not applicable.	
536872968	Computer1 (@3%s@): @1%s@ (@2%s@) No connection to the logging system backend	Not applicable.	

ID	Alarm text	Effect/causes	Solution
536872969	Computer1 [@3%s@] - @1%s@: An overflow of the transaction buffer and write buffer has occurred in log @2%s@ of the passive partner. Data may be lost. Avoid a manual failover of the redundancy partner while this alarm is active.	Not applicable.	
536872970	Computer1 [@3%s@] - @1%s@: (@2%s@) Connection to the long-term logging system is lost.	Not applicable.	
536872971	Computer1 [@4%s@] - @1%s@ (@2%s@ - @3%s@): Long-term logging system is out of hard disk space.	Not applicable.	
536872972	Computer1 [@3%s@] - @1%s@: (@2%s@): Long-term logging system no longer has any system resources.	Not applicable.	
536872973	Computer1 [@2%s@] - @1%s@: The passive and the active long-term logging system are no longer synchronized.	Not applicable.	

ID	Alarm text	Effect/causes	Solution
536872974	Computer1 [@2%s@] - @1%s@: Synchronizing long-term logging system	Not applicable.	
536872975	Computer1 [@2%s@] - @1%s@: Failover error in the long-term logging system.	Not applicable.	
536872976	Computer1 [@3%s@] - @1%s@: (@2%s@) A general error has occurred. Check the traces for additional information.	ContextLogging service: An unknown error occurred during the execution of a database operation.	Analyze previous traces to learn more about the problem. A possible cause can be the unexpected removal of a storage medium. In this case, reinstall the storage medium and restart Runtime. Otherwise, contact Siemens support for assistance.
536872977	Computer1 [@6%s@] - @1%s@ (@2%s@ - @3%s@): Logging medium cannot be reached. @5%t#4T@	ContextLogging service: When trying to create a database or reconnect to the storage medium, no storage medium was available or authorizations were lacking.	Check if the configuration (path of the log) is valid for all logs and if all required storage media are connected and working. If authorizations are available, ensure that the file system authorizations are set correctly.
536872978	Computer1 [@2%s@]: The logging system @1%s@ has detected a more recent database version that is not compatible with the current runtime version. Update the runtime version on your device or use a device with the same runtime version as the database version.	ContextLogging service: The Runtime software used is older than the databases used. Probably the database was moved from a device with newer software to a device with older software.	Move the database to a device with the same or a newer version of the Runtime software or update the software version of the device.

ID	Alarm text	Effect/causes	Solution
536872979	Computer1 [@2%s@]: @1%s@: Synchronizing the logging data of the redundant partners.	The redundant ContextLogging services synchronize the data with each other. This is normal behavior that only indicates that the historical data on both hosts is not quite up to date.	No action is required, but as long as this system event persists, it is recommended not to shut down any of the hosts. If this alarm has persisted since the last start-up/restart of one of the hosts, do not restart or shut down the longer running host or data loss may occur.
536872980	Computer1 [@2%s@]: @1%s@: Error during background synchronization. Check the traces for additional information.	ContextLogging service: An error occurred during data synchronization.	Check the additional information of the alarm to identify and eliminate the cause of the error. The synchronization is repeated at regular intervals so that it can be performed shortly after the cause has been eliminated.
536873217	Computer1 [@3%s@]: @1%s@ (@2%s@): has no connection to the logging system.	AuditTrail service has its lost connection to the database. Possible cause: • Internal database error. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the database server. Start/restart database. The log files of the database can also help to find the cause.
536873218	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Log is full.	At least one storage medium used by the AuditTrail service is full, so new value changes cannot be saved. If this situation continues for too long, the buffers for values may become full, resulting in data loss.	Check the status of the connected storage media and free up some space on the devices (almost) full. Changing the log configuration can help reduce the amount of disk space needed by limiting the maximum amount of space a particular log can use before deleting the data.
536873219	Computer1 [@4%s@]: @1%s@ (@2%s@ - @3%s@): Insufficient resources of the logging system.	Not applicable.	
536873220	Computer1 [@3%s@]: @1%s@: A write buffer overflow has occurred in log @2%s@. Data will be lost.	In the AuditTrail service, the buffers are full and therefore some data has already been lost. Possible cause: • Infrastructural problem (e.g. connection lost, hard disk full, see other system events in this case). • Generally, a high load that cannot be handled by the system. With redundant deployment, if only one host has this problem temporarily, the data from the other host is synchronized once the problem is resolved.	Correct the root cause of the problem. If the problem is infrastructural, follow the instructions in the appropriate system event. If there is none, the cause is probably too much load on the system or write speed degradation (for example, slowing down of a storage medium). Check the hardware elements, reduce the system load, or if it is a persistent problem, the hardware configuration may not be sufficient for the current project.

ID	Alarm text	Effect/causes	Solution
536873221	Computer1 [@2%s@]: @1%s@: The passive logging system has lost the connection to the active logging system.	Not applicable.	
536873222	Computer1 [@2%s@]: @1%s@: Synchronizing data	Not applicable.	
536873223	Computer1 [@2%s@]: @1%s@: Failover error	Not applicable.	
536873224	Computer1 (@3%s@): @1%s@ (@2%s@) No connection to the logging system backend	Not applicable.	
536873225	Computer1 [@3%s@] - @1%s@: An overflow of the transaction buffer and write buffer has occurred in log @2%s@ of the passive partner. Data may be lost. Avoid a manual failover of the redundancy partner while this alarm is active.	Not applicable.	
536873226	Computer1 [@3%s@] - @1%s@: (@2%s@) Connection to the long-term logging system is lost.	Not applicable.	

ID	Alarm text	Effect/causes	Solution
536873227	Computer1 [@4%s@] - @1%s@ (@2%s@ - @3%s@): Long-term logging system is out of hard disk space.	Not applicable.	
536873228	Computer1 [@3%s@] - @1%s@: (@2%s@): Long-term logging system no longer has any system resources.	Not applicable.	
536873229	Computer1 [@2%s@] - @1%s@: The passive and the active long-term logging system are no longer synchronized.	Not applicable.	
536873230	Computer1 [@2%s@] - @1%s@: Synchronizing long-term logging system	Not applicable.	
536873231	Computer1 [@2%s@] - @1%s@: Failover error in the long-term logging system.	Not applicable.	
536873232	Computer1 [@3%s@] - @1%s@: (@2%s@) A general error has occurred. Check the traces for additional information.	AuditTrail service: An unknown error occurred during the execution of a database operation.	Analyze previous traces to learn more about the problem. A possible cause can be the unexpected removal of a storage medium. In this case, reinstall the storage medium and restart Runtime. Otherwise, contact Siemens support for assistance.

ID	Alarm text	Effect/causes	Solution
536873233	Computer1 [@6%s@] - @1%s@ (@2%s@ - @3%s@): Logging medium cannot be reached. @5%t#4T@	AuditTrail service: When trying to create a database or reconnect to the storage medium, no storage medium was available or authorizations were lacking.	Check if the configuration (path of the log) is valid for all logs and if all required storage media are connected and working. If authorizations are available, ensure that the file system authorizations are set correctly.
536873234	Computer1 [@2%s@]: The logging system @1%s@ has detected a more recent database version that is not compatible with the current runtime version. Update the runtime version on your device or use a device with the same runtime version as the database version.	AuditTrail service: The Runtime software used is older than the databases used. Probably the database was moved from a device with newer software to a device with older software.	Move the database to a device with the same or a newer version of the Runtime software or update the software version of the device.
536873235	Computer1 [@2%s@]: @1%s@: Synchronizing the logging data of the redundant partners.	The redundant AuditTrail services synchronize the data with each other. This is normal behavior that only indicates that the historical data on both hosts is not quite up to date.	No action is required, but as long as this system event persists, it is recommended not to shut down any of the hosts. If this alarm has persisted since the last start-up/restart of one of the hosts, do not restart or shut down the longer running host or data loss may occur.
536873236	Computer1 [@2%s@]: @1%s@: Error during background synchronization. Check the traces for additional information.	AuditTrail service: An error occurred during data synchronization.	Check the additional information of the alarm to identify and eliminate the cause of the error. The synchronization is repeated at regular intervals so that it can be performed shortly after the cause has been eliminated.

Archiving data 7

7.1 Log basics

Introduction

The following types of logs are available for HMI devices:

- Data log
  A data log is used to log process data from an industrial plant. You can find more detailed information about this in section Logging tags (Page 653).
- Alarm log
  An alarm log is used to log alarms that occur in the monitored process. You can find more detailed information about this in section Logging alarms (Page 769).
- Context log
  The context log is used to save user-defined and system-generated contexts of plant objects. The context log is only available for Unified PC and is stored according to the settings in the "WinCC Unified Configuration" tool.
  The context log is not segmented and has no limit with regard to size or time period.

Database types

The following database types are supported by various HMI devices:

HMI device	Supported database type
SIMATIC Unified Comfort Panel	SQLite
SIMATIC WinCC Unified PC	SQLite
	Microsoft SQL

Note

Microsoft SQL for Unified PC

- Unified PCs use SQLite as the default database type. To use Microsoft SQL, the system provides an installation option with a setup package. Logging with SQLite is not possible after the installation of Microsoft SQL.
  Existing SQLite files are retained, but they cannot be accessed in runtime.
- For SQLite, the segmented backup is not supported.

Database for simulation

SQLite is always used for simulations on the Unified PC. Exception: Runtime is installed on the same Unified PC and Microsoft SQL is configured for runtime. The configuration of the storage location is ignored during the simulation. Instead, a relative path is used in the project folder.

Protection against loss of data

When the database cannot be reached for brief periods, a temporary buffer of 8 MB is available. The buffer is used, for example, during a system overload caused by large amounts of data or when changing the storage medium.

If the power supply is unintentionally interrupted, runtime is suddenly and unexpectedly shut down. In this case, a loss of data may occur for up to 4 seconds. If the system was overloaded by large amounts of data, the loss of data can be even higher.

The data already logged is retained and is available unchanged and completely after the power supply has been restored and the system has been rebooted. Logging can be continued with the same database.

NOTICE
Loss of data due to interrupted power supply and removed storage medium Loss of data will occur when the power supply is interrupted and there is no connection to the storage medium at the same time.

Protection against manipulation

Logs can contain sensitive and confidential content, such as performance parameters or product data, that must be protected from unintentional or unauthorized modification.

In addition to physical protection measures, such as barriers, the Unified PC logs can also be protected using standard tools such as access rights for folders.

On Microsoft SQL servers, you protect the log databases by using the Windows group "Simatic HMI" (read/write) and "Simatic HMI Viewer" (read). Only members of these groups have direct access to the databases.

Startup behavior

The startup behavior of the logs at runtime start is defined in the "Reset Logs" section of the "Load Preview" dialog box when the project is loaded.

The following options are available:

- "No reset": Existing logged data is retained. This setting adds the data to be logged to an existing log.
- "Reset all": Existing logged data is deleted.

Restoring log segments

You have the option of restoring log segments of tag and alarm logs in runtime with the Runtime Manager. You can visualize the restored data in a trend control, for example.

You can find more information on this in the help of the Runtime Manager.

See also

Specifying runtime settings (Page 7134)

Specifying runtime settings (Page 7170)

7.2 How it works

Introduction

Segmented logs are used for logging.

Each log consists of a configurable number of segments. The segments are filled one after the other.

When the maximum size or the maximum period of a log is reached, the oldest segment is deleted.

When the maximum size or the maximum period of a segment is reached, a new segment is created. The newly created segment is filled further.

Size of a log

You have the option of defining the size of a log both time-dependent and disk space dependent:

- Log time period: When the maximum period is reached, the oldest segment is deleted.
  If you specify the value "0" for "Log time period", the oldest segment is deleted when the maximum log size in megabytes is reached.
  The limitation of the log by a time period is thus disabled.
- Maximum log size in megabytes: When the maximum log size is reached, the oldest segment is deleted.
  If you specify the value "0" for "Maximum log size (MB)", the oldest segment is deleted when the maximum time period of the log is reached.
  The limitation of the log by log size is thus disabled.

When you define both properties, the oldest segment is deleted as soon as the defined period or the maximum log size has been reached.

Note

Performance problems with disabled segmentation

If you specify the value "0" for the time period of the log and the maximum log size, the value for "Single segment time period" and for "Maximum segment size in megabytes" must also be "0". The segmentation is thus disabled. All data is written to one segment. Disabled segmentation results in performance problems when accessing the log. You cannot create a backup when segmentation is disabled. Log contents can only be deleted with the system functions "ClearTagLog" or "ClearAlarmLog".

The database can grow uncontrolled, as the log is not limited in size and time period.

Define at least the log period or the maximum log size in megabytes.

When you define logs, keep the following rules in mind:

- The time period of a segment must not exceed the time period of the log.
- The maximum size of the segment must not exceed the maximum size of the log.

Note

Recommended log size

The following limits are recommended for the log size:

- At least 3 individual segments
- Max. 5000 segments (database type "Microsoft SQL")
- When the database type "SQLite" is used, the maximum number of segments is only limited by the underlying file system.

Backups and segments that were restored from backups with the Runtime Manager are not taken into account when the system checks the database size.

Size of a segment

According to the log, you can define the size of a segment both time-dependent and disk space dependent:

- Segment time period: When the segment reaches the specified time, the current segment is closed and a new segment is created and filled with data.
  When you specify the value "0" for the "Single segment time period", the single segment is filled until the maximum segment size has been reached. Time-based segmenting is thus disabled.
- Maximum segment size in megabytes: When the segment reaches the specified size, the current segment is closed and a new segment is created and filled with data.
  When you specify the value "0" for the "Maximum segment size (MB)", the single segment is filled until the maximum single segment time period has been reached. Size-based segmentation is thus disabled.

When you define both properties, the current segment is closed and a new segment is created as soon as the defined time or the maximum segment size has been reached.

Note

Performance problems with disabled segmentation

When you specify the value "0" for the "Segment time period" and for "Maximum segment size in megabytes", segmentation is disabled. All data is written to one segment. Disabled segmentation results in performance problems when accessing the log. You cannot create a backup when segmentation is disabled. Log contents can only be deleted with the system functions "ClearTagLog" or "ClearAlarmLog".

For SQLite, the segment size of a log is always an integer multiple of 4 MB. This means the actual segment size is 8 MB when you configure a segment size of 7 MB.

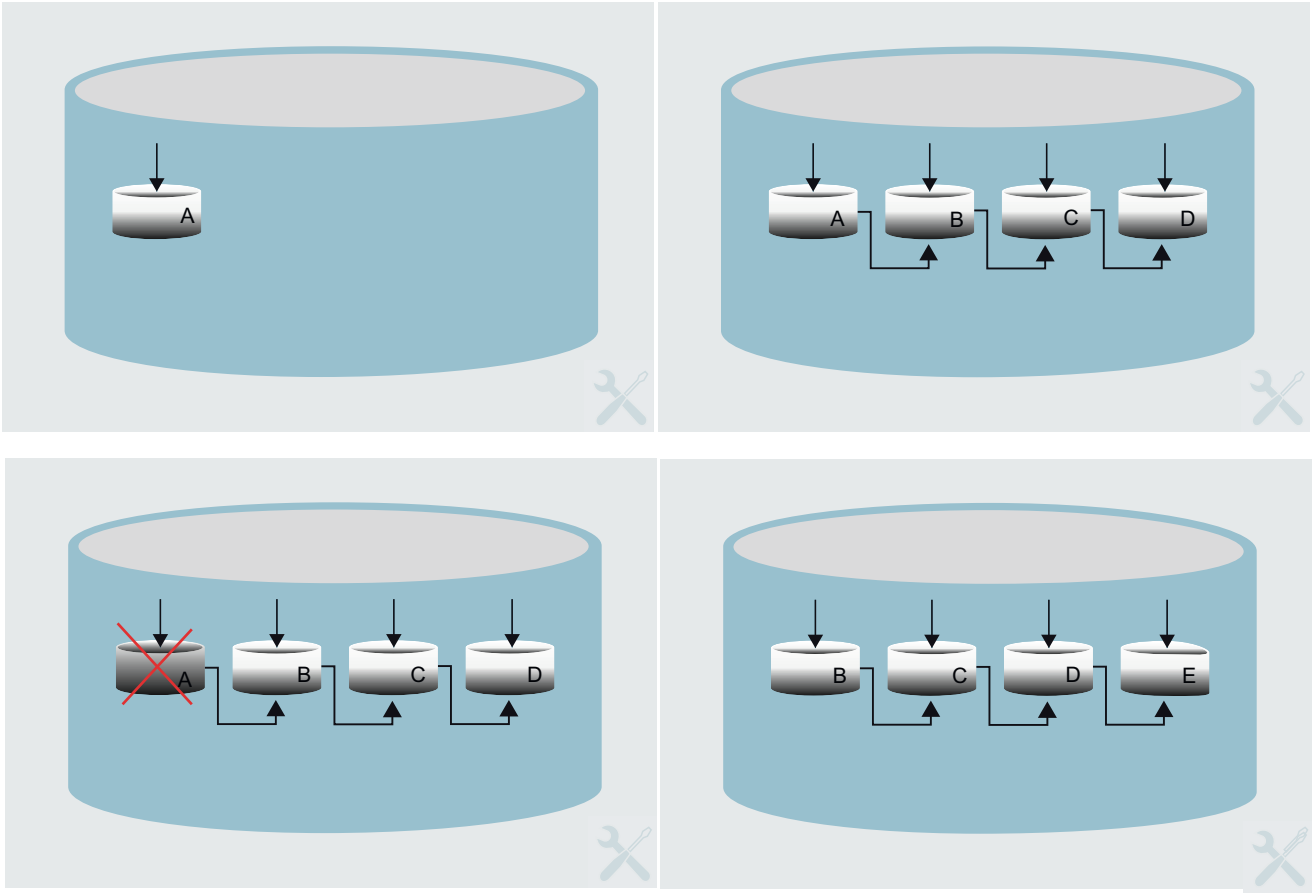
For Microsoft SQL, the smallest segment size is 3 MB.

Segment start time

You define the segment start time. The start time defines the time as of which log segments are written to the log.

Response to segment change

The individual segments are filled one after the other in runtime. Once a segment is totally full, the next segment is created and filled.



1. The process values are written continuously to the first segment.
2. When the configured size of the segment is reached or the period is exceeded, a new segment is created and filled.
3. When the maximum log size or the maximum period of a log is reached, the oldest segment is deleted.

To avoid losing process data as a result of deleting, you can configure a backup for the log.

Note

If you change the segmentation settings and load the project in runtime, a new segment is created.

Example

The following information has been configured:

Property	Value
Log time period	1 week
Maximum log size (MB)	700 MB
Segment time period	1 day
Maximum segment size (MB)	100 MB
Segment start time	Friday, 23 May 2020 18:00

With the configuration suggested in the table, log segments are written to the log starting on 23 May 2020 at 18:00 hours.

The next time-dependent segment changes take place cyclically after one day as of the configured time. The segment will also change if the configured size of 100 MB is exceeded in the course of one day.

When the maximum log size of 700 MB or the configured time period of one week is reached, the oldest segment is deleted.

Backup

You can swap out data from the log database as a backup with the database type "Microsoft SQL".

When you create a log you can configure a backup.

The data is swapped out in segments. A segment is always swapped out during a segment change when a new segment is started.

The larger the segment, the more time a backup requires. When the backup of a segment is complete, you can delete the segment.

Note

Creating backups

The backup is created with a delay of approx. 10 minutes.

If data is changed in a segment for which a backup has already been created, a new backup will be created approx. 10 minutes later.

See also

Log basics (Page 837)

Creating a data log and an alarm log (Page 848)

Size of a log entry in the data log (Page 654)

Size of a log entry in the alarm log (Page 770)

7.3 Storage locations of logs

Introduction

You adapt the main database storage locations for tag and alarm logging in the runtime settings of the HMI device. Different options are available depending on the HMI device.

For Unified PCs, you have the option of storing the storage location in the "WinCC Unified Configuration" tool under "Archive Settings".

Note

Default setting for the database storage location

The storage location "Default" is configured as default setting in the Runtime settings and in the "Logs" editor. If you do not change these options, the database is saved according to the setting in the "WinCC Unified Configuration" tool.

Storage location of a log

You adjust the storage location of the main database of the tag logs and alarm logs in the runtime settings of the HMI device.

The following options are available depending on the HMI device:

HMI device	Supported main database storage locations for logging
Unified Comfort Panel	• SD-X51 Many read and write processes are being executed during logging. Therefore, you should use memory cards rather than USB flash drives. • USB-X61 • USB-X62 • Off: Logging is disabled. You have the option of specifying a path on the storage medium.
Unified PC	• Default: In the "WinCC Unified Configuration" tool, you store the path at which the logs are saved under "Archive Settings". • Local: Select a path on the local file system, on an external storage medium or a network folder. We do not recommend using network folders, because the power supply can be interrupted at any time. • Project folder: The logs are saved in a subfolder of the Runtime project folder. • Off: Logging is disabled.

When you define a storage location in the runtime settings of the HMI device, and thus enable logging, you can specify the database storage location for individual tag and alarm logs in the "Logs" editor under "Storage medium".

It must be ensured that the "WCCILScsService" service user under which Runtime is running has read and write rights to the directory in which the logs are stored. You have the following options here:

- Configure the directory for the logs with the "WinCC Unified - Configuration" tool and configure the storage of logs in this directory or its subdirectories in the engineering system.
- In the engineering system, configure the storage of logs in directories for which no restriction of the access rights is configured in the operating system. However, this means that the logs are then not protected from access by third parties.

Note

If you want to access a large number of log values within a short time, it is advisable to set the maximum memory of the SQL server to at least 4 GB in the "WinCC Unified – Configuration" tool.

Changing the storage location of the main database

Note

Data loss when changing the location of the main database

If you have already saved data in a log and then change the location of the main database and reload the project, you can no longer access the already logged data in runtime.

Changing the location of individual logs in the "Logs" editor is possible without data loss as long as the location of the main database remains unchanged.

In the following cases, access to already logged data is lost:

- Unified PC:
  - If you use the storage medium "Standard" as the storage location of the main database in the runtime settings of the Unified PC and change the storage location of the databases in the Werkzeug "WinCC Unified - Configuration" tool
  - If you use the storage medium "Local" as the storage location of the main database in the runtime settings of the Unified PC and change the folder
  - If you change the file path of the log in the file system of the Unified PC, e.g. rename folders with a file explorer
- Unified Comfort Panel:
  - If you change the storage medium or folder of the main database location in the runtime settings of the Unified Comfort Panel
  - If you change the file path of the log on the external storage medium, e.g. rename folders with a file explorer

Naming conventions

The log names must be unique in an HMI device. Even if different storage locations are selected for different logs, the log name must be unique.

Syntax examples for storage locations

Location of local file system on the Unified PC:

- <C:\My_File_Folder\My_Archives\Machine_1>: Saves the log on the local hard disk drive "C:" in the subdirectory "My_File_Folder\My_Archives\Machine_1"

Memory card storage location on the Unified Comfort Panel:

- </media/simatic/data-storage /My_Archives/TagLogs>: Saves the log in the subdirectory "My_Archives\TagLogs"

File name

The segments are stored according to the syntax <Unique name of the runtime project>_<Abbreviation of the logging service>_<System ID>_<Start date of the segment>_<Start time of the segment (UTC)> for example, as follows:

- Microsoft SQL: "HMI_RT_1_TLG200_20201106_135307.mdf" and "HMI_RT_1_TLG200_20201106_135307_log.ldf"
- SQLite: "HMI_RT_1_TLG200_20201106_135307.db3"

When you use the "Microsoft SQL" database type, you can create a backup. The backup files of the segments are stored with the syntax <Unique name of the Runtime project>_<Abbreviation of the logging service>_<System ID>_<Start date of the segment>_<Start time of the segment (UTC)> under the configured path, for example as follows:

- "HMI_RT_1_TLG200_20201106_135307.bak"
- "HMI_RT_1_TLG200_20201106_135307_diff.bak"
  The file with the extension "_diff.bak" is created when the backup was already created and value changes occurred later, for example, by manually adding values.

External storage medium

You have the option of saving the logs on an external storage medium. To avoid the loss of data, the storage medium must be properly removed.

NOTICE
Data loss due to improper removal of the storage medium When the storage medium is not removed correctly, logging is interrupted. As soon as you connect a storage medium and restart runtime, logging can be continued with existing databases. Data that occur between improper removal of the storage medium and the start of runtime are not logged.

The procedure for ejecting the storage medium differs depending on the HMI device:

- Unified Comfort: You use the system function "EjectStorageMedium".
- Unified PC: Eject the storage medium using the PC operating system.

Next, you remove the storage medium and connect a different storage medium.

Data that occurs during the change process are written to a buffer. Logging is continued in a new segment after the change has been completed.

Logs on different storage media

Note

Storage medium for Unified Comfort

The storage location of the main database of a logging type and the configured storage medium of all associated logs must be identical.

Tag persistency

In runtime, you have the option of specifying tag persistency for internal tags. A separate database, in which the last values of the persistent tags are stored, is used for tag persistency.

When the databases for data logs or alarm logs and the database for tag persistency are stored on the same storage medium and the medium is changed while runtime is running, tag persistency can be affected.

Note

Use different storage media for tag persistency and logging.

Parameter set types

With Unified Comfort, you have the option of saving parameter set types on external storage media.

When the databases for data logs or alarm logs and the parameter set types are stored on the same storage medium and the medium is changed while runtime is running, the parameter set types can be affected.

Note

Use different storage media for parameter set types and logging.

See also

Storage system (Page 7138)

Storage system (Page 7173)

Basics of downloading projects (Page 7182)

Basics for downloading projects (Page 7147)

7.4 Creating a data log and an alarm log

Introduction

You create data logs and alarm logs for the HMI device in the "Logs" editor. In addition to the name, storage location and backup, you define the size of the log and its segments.

The size of the log and the segments can be defined both time-dependent and disk space dependent. The size of a log is calculated according to the log type as follows:

- Data log: Number of entries * Size of entries
  In addition to the value of a logging tag, its time stamp and quality code are logged.
- Alarm log: Number of entries times the approximate size of the entries
  The size of an entry depends on the alarm text, whether it contains parameter boxes and in which languages the alarm text is logged.

Note

Make sure that the log size does not exceed the free disk space and the system limits. The system does not validate the selected settings. A high number of linked log segments can lead to prolonged waiting periods in the system when starting and ending runtime.

Requirement

- An HMI device has been created.
- A database storage location for data logs and alarm logs is stored in the runtime settings.

Procedure

To create data logs or alarm logs, follow these steps:

1. Double-click the "Logs" entry in the project tree below the HMI device.
   The "Logs" editor opens.
2. Double-click "<Add>" in the "Name" column of the "Data logs" or "Alarm logs" tab.
   A new log is created.
3. Specify the name of the log.
   You can assign any name to the log. The name must contain at least one letter or one number.
   You can create several logs for each HMI device.
   The name must be unique for the respective HMI device.

4. Select the storage location of the log in the "Storage medium" field:
   – "Local": Enter the storage path for the log under "Storage directory".

   Note

   Logging on network drives

   Do not save databases directly on a network drive. Power supply can be interrupted at any time. This means there is no guarantee for reliable operation of logs.

   Save the logs on the local hard drive or an external storage medium, for example, a USB stick.

   In connection with SQLite, loss of data can occur for an external storage medium without journaling file system in case of an unintentional system crash.

   – "Default": Accepts the settings that you have defined in the runtime settings of the HMI device under "Storage system".

   Note

   Data loss when changing the location of the main database

   If you have already saved data in a log and then change the location of the main database and reload the project, you can no longer access the already logged data in runtime.

   Changing the location of individual logs in the "Logs" editor is possible without data loss as long as the location of the main database remains unchanged.

5. In the "Log time period" field, define the maximum time period for logging in the format <day>.<hour>:<minute>:<second>[.<millisecond>].
6. Define the maximum size in megabytes in the "Maximum log size (MB)" field.

   Note

   We recommend the following number of segments:
   - Without backup: 3 segments
   - With backup: 4 segments

7. In the "Segment" area, define the time period for a single segment in the format <day>.<hour>:<minute>:<second>[.<millisecond>].
8. Define the maximum segment size.

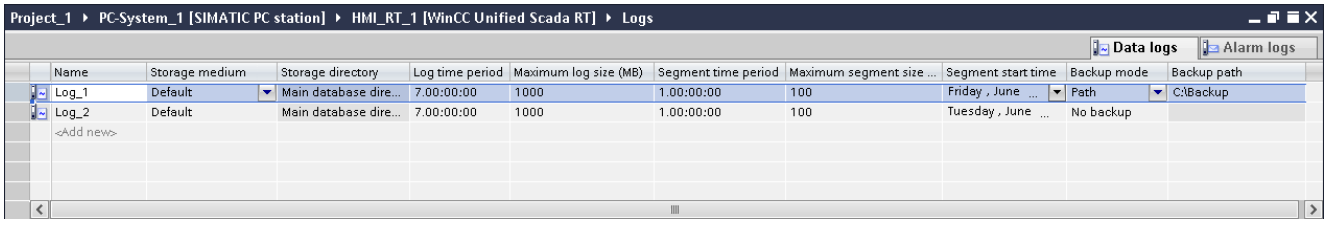
9. Define the start time.
   The start time defines the time as of which log segments are written to the log.
10. Set whether data is to be backed up and specify the path for the backup under "Backup > Backup mode".

Note

Configuring a backup

Backups can only be created when you use the "Microsoft SQL" database type. You configure the database type in the runtime settings of the HMI device.

The backup is created with a delay of approx. 10 minutes.
If data is changed in a segment for which a backup has already been created, a new backup will be created about 10 minutes later.
When you subsequently change the primary path, the new backup file is written to the new storage path after loading. The previous backup files remain in the previous storage location.



Tips for an efficient procedure

Configure the properties of a log directly in the "Logs" editor table. To view hidden columns, activate the column titles using the shortcut menu.

Result

- The log is created.
- The log database in the configured folder is created after the project is started in runtime.

See also

Configuring logging tags (Page 657)

Basics of alarm logging (Page 769)

How it works (Page 839)

7.5 Editing log contents with scripts and system functions

Using snippets

In the shortcut menu of the "Scripts" editor, you will find various snippets for logging. You have the following options, for example:

- Read data logs and alarm logs
- Export data logs and alarm logs

- Correct entries in data logs
- Comment entries in data logs

Delete log contents

You can delete the contents of an alarm log or data log using the "ClearTagLog" or "ClearAlarmLog" system functions. The log itself is retained, the alarm or logging data stored in it is deleted. This procedure can be useful, for example, after a test phase has been completed when existing logs are to be emptied.

See also

Alarm control (Page 309)

ClearTagLog (Page 934)

ClearAlarmLog (Page 933)

Input support (Page 982)

System functions (Page 909)

Configuring parameter sets

8

8.1 Basics

8.1.1 Basics of parameter control

Introduction

The parameter control is a comprehensive function for the control of parameter sets for configuration engineers, operators and recipe creators. The parameter control brings you the following benefits:

- You can apply the structure of a user data type for one or more parameter set types.
- You can change the structure of one or more parameter set types automatically via a new user data type version.
- You can exchange a large number of parameters manually or automatically between HMI device and PLC to set up a machine for a production.
- You can create parameter sets simply and uniformly for products to be manufactured in the works during engineering or during ongoing operation.
- By structuring the associated parameters/setpoints, you can easily transfer parameters.

Elements of the parameter control

- Parameter set type
  A parameter set type with parameter set type items determines the structure that is used for parameter sets on a machine. You create a parameter set type with parameter set type items on the basis of a released HMI or PLC user data type that has user data type elements.
- Parameter set type item
  Element of a parameter set type that is based on a user data type element. A parameter set type item has the same name and data type as the corresponding user data type element.
- Parameter record
  Set of parameters with concrete values that can be activated on a machine.
- Parameters
  Element of a parameter set that is based on a parameter set type item. A parameter has the same display name and data type as well as the same unit of measure as the corresponding parameter set type item. A parameter has a concrete value that can be activated on a machine.

Tools of the parameter control

- "Parameter set types" editor
  In the "Parameter set types" editor, you create parameter set type items on the basis of an HMI or PLC application data type. In addition, you configure the properties of parameter set types and parameter set type items in the editor.
- Parameter set control
  The parameter set control is a control with which you can display and manage parameter sets in runtime and exchange them with the PLC.

Parameter set memory

Part of the parameter control is the parameter set memory. The parameter set memory can be configured depending on the device.

Parameter control in the Engineering System

Perform the following tasks in Engineering System for the parameter control:

- You create a parameter set type to specify the structure of parameter sets.
- You change a parameter set type to change the structure of parameter sets.
- You assign a tag of the data type user data type to a parameter set type to transfer parameter sets between HMI device and PLC in runtime.
- You create local scripts in screen objects or tasks to transfer parameter sets in runtime between the HMI device and PLC.
- You assign control tags to a parameter set type to automatically transfer or delete parameter sets between HMI device and PLC in runtime.
- You configure a parameter set control to display parameter sets, manage them and exchange them with the PLC through the Control in runtime.
- In a screen, you configure an individual input mask to display, manage and exchange parameter sets with the PLC without using the control "Parameter set control".

Parameter control in runtime

The following options are available in runtime with the parameter control:

- You create, change and delete parameter sets in a parameter set control to manage parameter sets for different productions.
- Alternatively, create, change and delete parameter sets in an individual input mask to manage them for different productions.
- You export parameter sets from the parameter set memory into a "*.tsv" file to edit them in a text editor.

  Note

  A "*.tsv" file is a text file that uses the tabulator as a list separator.

- You import parameter sets from a "*.tsv" file into the parameter set memory.

- You transfer parameter sets manually or automatically to the control system to set up machines with values for different productions.
- You read parameter sets manually or automatically from the PLC to call up currently used values of production machines for later use.
- You automatically delete parameter sets from the parameter set memory.

See also

Configuring parameter sets (Page 863)

Using parameter sets in runtime (Page 885)

Configuring user data types (Page 645)

8.1.2 Limitations

Unsupported data types in the parameter data set

Structure data types are not supported in a parameter data set:

- Struct, Array of Struct
- ErrorStruct
- CREF
- NREF

Restrictions on the PLC user data type

Observe the following restrictions when creating the PLC user data type:

- The user data type may have a maximum of 1000 elements.
- No user data type item may have the data type Time_Of_Day.
- The ARRAY data type is supported.
  The use of a user data type is not supported in an ARRAY.
- The user data type may have a maximum of 8 levels.

8.1.3 "Parameter set types" editor

Introduction

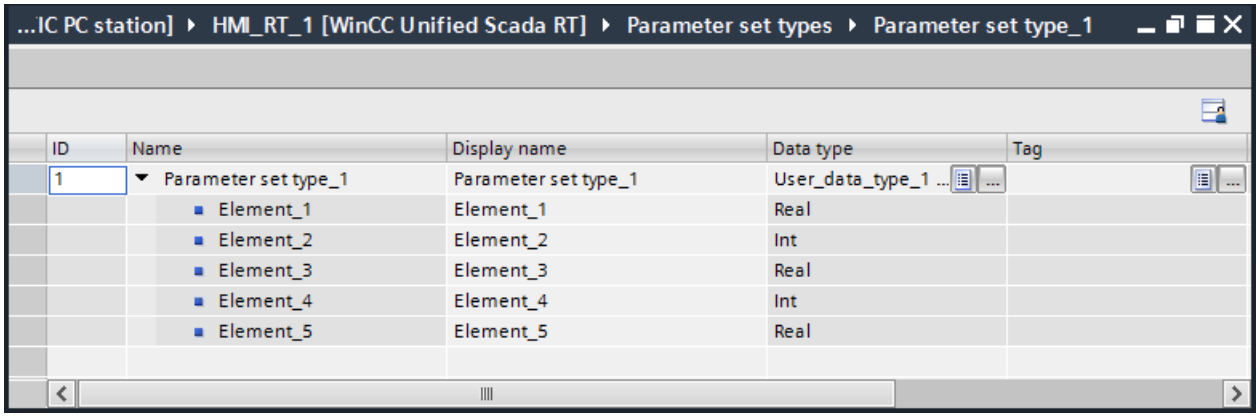
In the "Parameter set types" editor, you create a parameter set type with elements on the basis of an HMI or PLC application data type. In addition, you configure the properties of parameter set types and parameter set type items in the editor.

Note

Alternatively create the parameter set type items on the basis of a user data type in the Inspector window. In addition, you configure the properties of parameter set types and parameter set type items also alternatively in the Inspector window.

Structure of the "Parameter set types" editor

The "Parameter set types" editor is a tabular editor. The editor always contains only a single parameter set type and, if applicable, its elements. To view hidden columns, activate the column titles using the shortcut menu.



Properties of parameter set types

In the "Parameter set types" editor, you can configure the following properties:

Property	Description
ID	Number of a parameter set type. The ID uniquely identifies a parameter set type within the HMI device. The ID appears in the parameter set control in runtime.
Name	Name of a parameter set type. The name uniquely identifies a parameter set type within the HMI device.
Display name	Display name of a parameter set type. The display name appears in the parameter set control in runtime. You can configure display names in multiple languages. The property is optional. If you do not set a display name, the value from the "Name" property appears in the parameter set control in runtime.
Data type	Enabled HMI or PLC user data type with which you define the structure of a parameter set type.

Property	Description
Tag	External HMI tag of the data type HMI or PLC user data type. In runtime you transfer parameter sets between HMI device and PLC via the HMI tag.
Edit tag	Local session HMI tag used to manage the values of a parameter set in a screen. Edit tags can be read from or written to the PLC. The HMI tag must not contain any information about the range (Upper 2, Lower 2). The property is optional.
Parameter ID	Control tag with numerical data type. The control tag is used to define an ID of a parameter set which is the target of one of the following control jobs in runtime: • Control job with job ID "6": Reading a parameter set from the PLC and storing it in the parameter set memory. • Control job with job ID "7": Loading a parameter set from the parameter set memory and writing it to the PLC. • Control job with job ID "8": Deleting a parameter set from the parameter set memory. The property is optional.
Job ID	Control tag with numerical data type. The control tag is used to define a control job which is applied to a parameter set in runtime. In runtime, you can apply the following control jobs to a parameter set: • Control job with job ID "6": Reading a parameter set from the PLC and storing it in the parameter set memory. • Control job with job ID "7": Loading a parameter set from the parameter set memory and writing it to the PLC. • Control job with job ID "8": Deleting a parameter set from the parameter set memory. The property is optional.
Author	Author of a parameter set type. The property is optional. By default, the property is pre-assigned with the logged-on Windows user. By default, the column is hidden.
Version	Version of a parameter set type. The property is optional. By default, the property is pre-assigned with the date and time of creation of the parameter set type. By default, the column is hidden.
Storage medium	For Unified PC, the storage medium is internal. The parameter set type is saved in the system default directory. For Unified Comfort Panel, you choose between the internal storage medium and various external storage media.
Memory folder	Internal storage medium: System standard folder in runtime, in which the parameter set memory is located. The folder refers to the folder of the runtime project. The property is write-protected. External storage medium: Freely selectable memory folder.

Properties of parameter set type items

In the "Parameter set types" editor, you can configure the following properties for parameter set type items:

Property	Description
Name	Name of a parameter set type item. The name uniquely identifies a parameter set type item. The name is write-protected and identical to the name of the corresponding user data type element.
Display name	Display name of a parameter set type item and a corresponding parameter in a parameter set. The display name appears in the table of the parameter set control in runtime. You can configure display names in multiple languages. The property is pre-assigned with the value from the "Name" property.
Data type	Data type of a parameter set type item and a corresponding parameter in a parameter set. The name is write-protected and identical to the data type of the corresponding user data type element.
Start value	Start value of a parameter set type item. The start value is used to pre-assign a corresponding parameter in a newly created parameter set in runtime. If you have not set a start value in a user data type element with numerical data type or bit sequence data type, the corresponding parameter set type item receives "0" as the start value. When parameter set type items with the data type ARRAY are used, only the start value for the items of the array can be defined. The property is optional for parameter set type items with string data types.
Minimum value	Minimum permissible value of a parameter set type item and a corresponding parameter in a parameter set. The minimum value of a parameter set type item may not be below the minimum value of the data type of the parameter set type item. When parameter set type items with the data type ARRAY are used, only the minimum value for the items of the respective lowest level can be defined. The property is optional and is disabled for parameter set type items with string data types and bit sequence data types.
Maximum value	Maximum permissible value of a parameter set type item and a corresponding parameter in a parameter set. The maximum value of a parameter set type item may not be above the maximum value of the data type of the parameter set type item. When parameter set type items with the data type ARRAY are used, only the maximum value for the elements of the respective lowest level can be defined. The property is optional and is disabled for parameter set type items with string data types and bit sequence data types.

Property	Description
Value required	If the check box of the property "Value required" is selected in a parameter set type item, a corresponding parameter must have a value in a parameter set. Otherwise, a corresponding parameter must have no value in a parameter set. The check box of the property is selected by default for parameter set type items with numerical data types. In this case, you cannot clear the check box. The check box of the property is cleared by default for parameter set type items with string data types. In this case, however, you can select the check box if required. When parameter set type items with the data type ARRAY are used, only the check box for the items of the respective lowest level can be defined.
Unit of measure	Unit of measure of a parameter set type item and a corresponding parameter in a parameter set. The unit of measure appears in the table of the parameter set control in runtime. When parameter set type items with the data type ARRAY are used, only the unit of measurement for the items of the respective lowest level can be defined. The property is optional.

See also

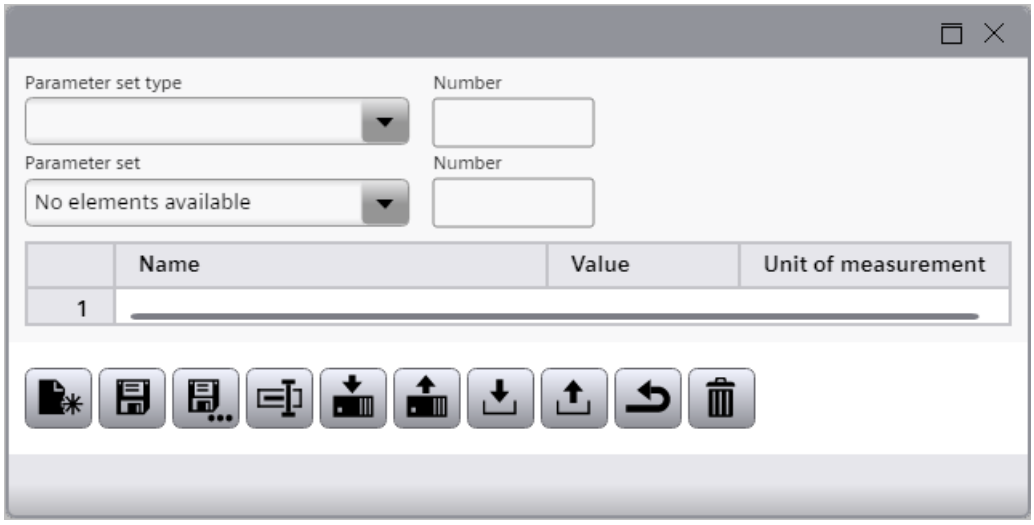
Creating a parameter set type with elements via an HMI user data type (Page 863)

Creating a parameter set type with elements via a PLC user data type (Page 866)

8.1.4 Parameter set control

Use

You can use the "Parameter set control" object to display and manage parameter sets in runtime and to exchange them with the controller.



Layout

You can change the settings for the position, geometry, style, color, and font of the object in the Inspector window. Under "Miscellaneous", adapt the following properties:

- "Editing mode": Defines the activation status of the toolbar buttons.
- "Toolbar": Defines the buttons of the parameter set control.
- "Information bar": Specifies the representation of the information bar.
- "Parameter view": Specifies the display of the parameter table in the object.

Using a parameter set type

If you only want to use a particular parameter set type with its parameter sets in runtime, select the desired parameter set type under "Properties > General > Fixed parameter set type".

Configuring the time zone

To configure the time zone, follow these steps:

Under "Properties > Miscellaneous > Time zone", set the desired time zone by entering a numerical value.

The numerical value stands for a time zone, for example:

- "-1" stands for UTC-1h (Central European Time, standard time)
- "1" stands for UTC-12h (International Date Line West)
- "2" stands for UTC-11h (Hawaii)

Defining the editing mode

To specify the editing mode and to enable or disable the buttons, follow these steps:

Under "Properties > Miscellaneous > Editing mode", configure the activation status of the toolbar buttons "Create", "Save", "Save as", "Rename" and "Delete". These toolbar buttons are used to edit parameter sets.

You can select between the following settings:

- "None": Deactivates all buttons.
- "Update": Activates the "Save" and "Rename" buttons.
- "Create": Activates the "Create" and "Save as" buttons.
- "Delete": Activates the "Delete" button.

Configuring reordering of the columns

Configure whether operators can reorder the table columns in runtime using drag-and-drop. More information is available in the section Configuring reordering of the columns (Page 392).

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Configuring the information bar

To configure the information bar, follow these steps:

1. Configure the general properties of the information bar, such as the font and background color, under "Properties > Miscellaneous > Information bar".
2. To adjust the height of the "Status text" element, specify the height under "Properties > Miscellaneous > Information bar > Elements > [0] Element".

The "Status Text" element is the only status line element of the parameter set control. Status messages are displayed in this element in runtime.

Toolbar

You can define the buttons of the parameter set control in runtime and their operator authorizations in the Inspector window under "Properties > Miscellaneous > Toolbar > Elements". By default, all buttons are displayed in the toolbar. To hide specific buttons, deactivate the "Visibility" property in the settings of the corresponding button.

"Toolbar": Defines the buttons of the parameter set control.

Note

Visibility of buttons in runtime

Buttons for which the "Visibility" option is deactivated during the configuration cannot be made visible again in runtime by a script. After loading into a device, the array of available items contains only those buttons that are configured as visible.

The following buttons are available for the parameter set control:

	Button	Function
	Create	Creates a new parameter set.
	Save	Saves a parameter set.
	Save as	Saves an existing parameter set under a new name and new ID.
	Rename	Renames the selected parameter set.
	Write to PLC	Writes the values of the selected parameter set to the PLC.
	Read from PLC	Writes the values of the selected parameter set from the PLC.
	Import	Imports parameter sets from a "*.tsv" file.
	Export	Exports parameter sets to a "*.tsv" file.
	Cancel	Cancels the process.
	Delete	Deletes the selected parameter set.

Note

A "*.tsv" file is a text file that uses the tabulator as a list separator.

See also

Configuring the parameter set view (Page 880)

8.2 Configuring parameter sets

8.2.1 Creating a parameter set type with elements via an HMI user data type

Introduction

You have created an HMI user data type with elements. You can assign the user data type to one or more parameter set types and this way apply the elements of the user data type for the parameter set type or types. A parameter set type with elements determines the structure that is used for parameter sets on a machine. Because you do not have to create the elements of the user data type again separately for the parameter set type(s), efficiency and reusability are a given.

Note

Observe the following restrictions when creating the HMI user data type:

- The user data type may have a maximum of 1000 elements.
- No user data type item may have the data type Textref, Time_Of_Day, or Raw.

Note

You can also create a parameter set type with elements via a PLC user data type that is created in a SIMATIC S7-1200/1500 control system.

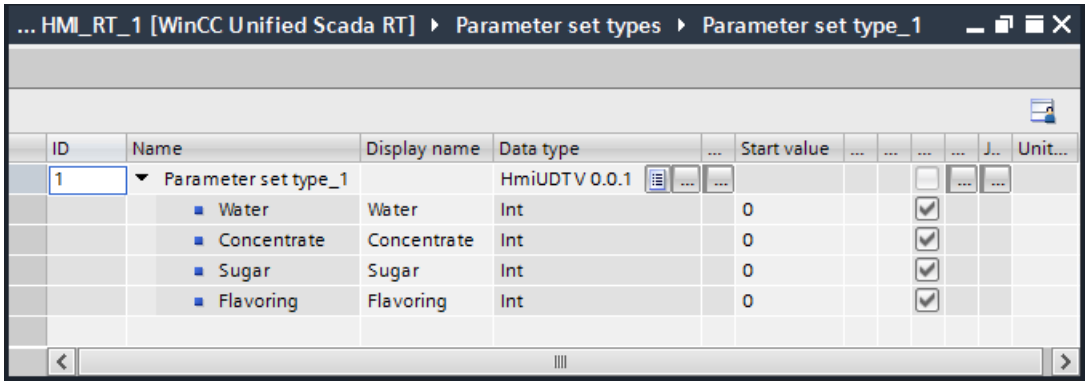
Requirement

- An HMI device has been created.
- An HMI user data type with a user data type element is created.
- The communication driver SIMATIC S7-300/400 or SIMATIC S7-1200/1500 is set for the HMI user data type.
- The HMI user data type is released.

Creating a parameter set type with elements via an HMI user data type

Proceed as follows to create a parameter set type with elements via an HMI user data type:

1. Open the "Parameter set types" folder in the project tree of the HMI device.
2. Double-click "Add new parameter set type".
   A parameter set type with unique standard name and unique ID is created. The "Parameter set types" editor opens.
3. Select the released HMI user data type in the "Data type" column in the "Parameter set types" editor.
   In the editor, parameter set elements which are based on the user data type elements are added to the parameter set type.



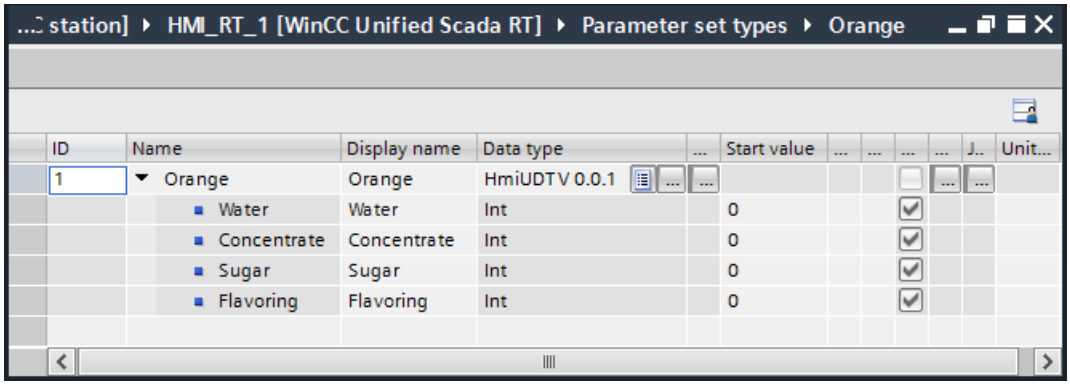
Note

The parameter set type items have the same name and data type as the user data type elements. The name and data type of the parameter set type items are write-protected in the editor.

Configuring a parameter set type

To configure the created parameter set type, proceed as follows:

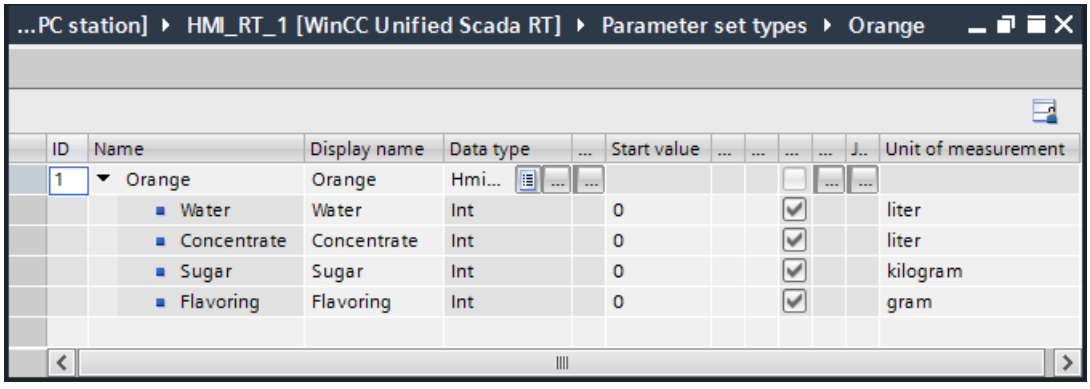
1. Enter a meaningful name for the parameter set type in the "Name" column in the "Parameter set types" editor
   The name uniquely identifies the parameter set type within the HMI device.
2. Enter a language-dependent name for the parameter set type in the "Display name" column.
   The display name appears in the parameter set control in runtime.
3. If required, enter an own number for the parameter set type in the "ID" column.
   The ID uniquely identifies the parameter set type within the HMI device. The ID appears in the parameter set control in runtime.



Configuring parameter set type items

To configure the created parameter set type items, proceed as follows:

1. In the "Display name" column of the "Parameter set types" editor configure a language-dependent name for the parameter set type items and corresponding parameters in a parameter set.
   The display name appears in the table of the parameter set control in runtime.
2. Configure a unit of measure In the "Unit of measure" column for parameter set type items and corresponding parameters in a parameter set.
   The unit of measure appears in the table of the parameter set control in runtime.
3. Configure a start value in the "Start value" column for parameter set type items.
   The start value is used to pre-assign the corresponding parameters in a newly created parameter set in runtime.



Result

You have created a parameter set type with elements via an HMI user data type.

See also

Creating an HMI user data type (Page 645)

Creating HMI user data type elements (Page 647)

"Parameter set types" editor (Page 856)

Changing a parameter set type with elements (Page 870)

Configuring the parameter set view (Page 880)

Assigning a tag of the data type HMI user data type to a parameter set type (Page 873)

8.2.2 Creating a parameter set type with elements via a PLC user data type

Introduction

You have created a PLC user data type with elements. You can assign the user data type to one or more parameter set types and this way apply the elements of the user data type for the parameter set type or types. A parameter set type with elements determines the structure that is used for parameter sets on a machine. Because you do not have to create the elements of the user data type again separately for the parameter set type(s), efficiency and reusability are a given.

Note

Observe the following restrictions when creating the PLC user data type:

- The user data type may have a maximum of 1000 elements.
- No user data type item may have the data type Time_Of_Day.
- The ARRAY data type is supported.
  The use of a user data type is not supported in an ARRAY.
  The user data type may have a maximum of 8 levels.

Note

You can also create a parameter set type with elements via an HMI user data type for which the SIMATIC S7-300/400 or SIMATIC S7-1200/1500 communication driver is set.

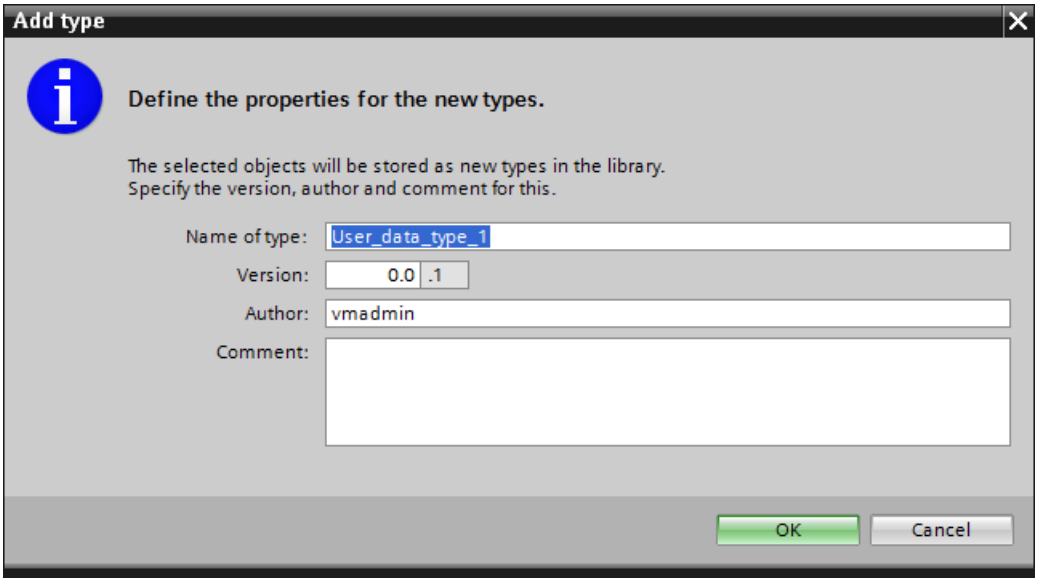
Requirement

- The HMI device has been created.
- A PLC user data type with user data type elements is created in a SIMATIC S7-1200/1500 PLC.
- The "Libraries" task card is open.
- The project library is open.

Adding a PLC user data type to the project library

To add a PLC user data type to the project library, follow these steps:

1. Open the PLC's folder in the project tree.
2. Open the folder "PLC data types" in the folder of the controller.
   The created PLC user data type is displayed in the "PLC data types" folder.
3. Move the PLC user data type to the "Types" folder in the project library.
   The "Add type" dialog opens.



4. Make the following settings in the dialog:
   - Enter a unique name for the PLC user data type under "Type name".
   - Enter a valid version number for the PLC user data type under "Version".
   - Enter the author responsible for the PLC user data type under "Author".
   - Enter a comment for the PLC user data type under "Comment".
5. Click the "OK" button.
   The PLC user data type is added with a released version to the "Types" folder in the project library.

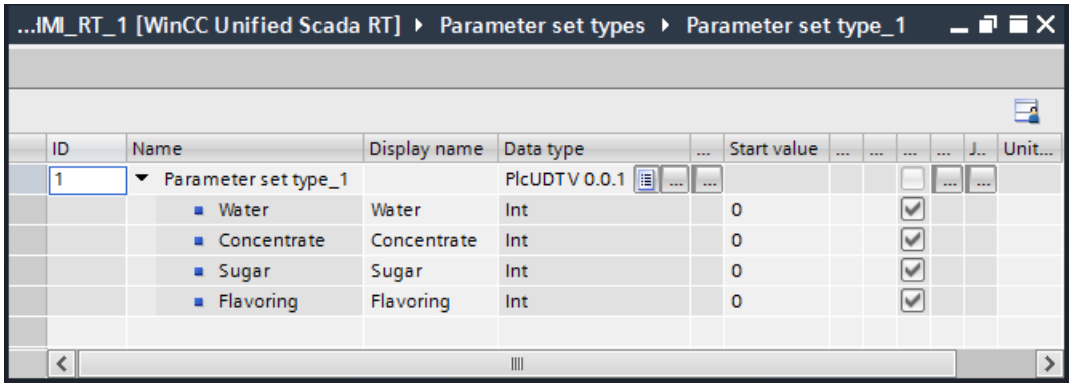
Note

When you use other user data types within the user data type that are not yet available in the library, these user data types are also applied to the library.

Creating a parameter set type with elements via a PLC user data type

Proceed as follows to create a user data type with elements via a PLC user data type:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click "Add new parameter set type".
   A parameter set type with unique standard name and unique ID is created. The "Parameter set types" editor opens.
3. Select the released PLC user data type in the "Data type" column in the "Parameter set types" editor.
   In the editor, parameter set elements which are based on the user data type elements are added to the parameter set type.



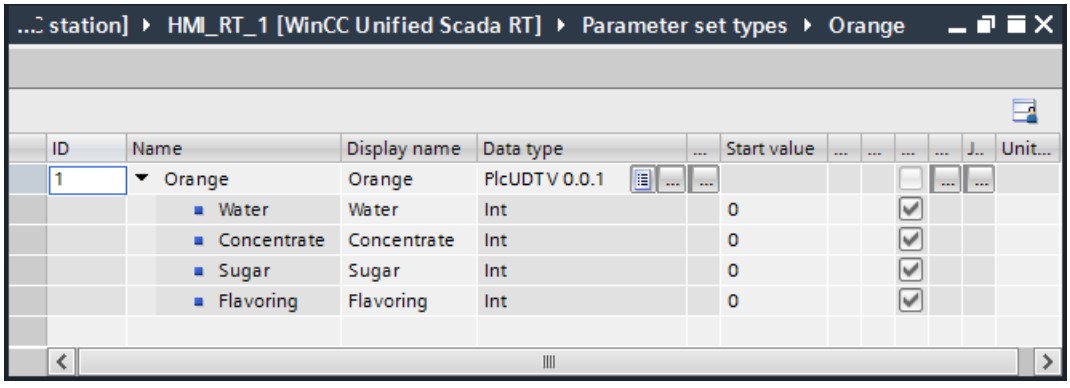
Note

The parameter set type items have the same name and data type as the user data type elements. The name and data type of the parameter set type items are write-protected in the editor.

Configuring a parameter set type

To configure the created parameter set type, proceed as follows:

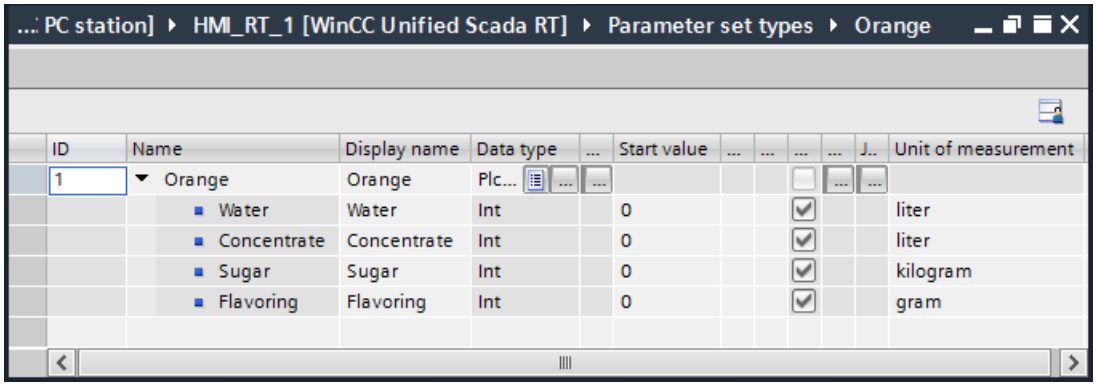
1. Enter a meaningful name for the parameter set type in the "Name" column in the "Parameter set types" editor
   The name uniquely identifies the parameter set type within the HMI device.
2. Enter a language-dependent name for the parameter set type in the "Display name" column.
   The display name appears in the parameter set control in runtime.
3. If required, enter an own number for the parameter set type in the "ID" column.
   The ID uniquely identifies the parameter set type within the HMI device. The ID appears in the parameter set control in runtime.



Configuring parameter set type items

To configure the created parameter set type items, proceed as follows:

1. In the "Display name" column of the "Parameter set types" editor configure a language-dependent name for the parameter set type items and corresponding parameters in a parameter set.
   The display name appears in the table of the parameter set control in runtime.
2. Configure a unit of measure In the "Unit of measure" column for parameter set type items and corresponding parameters in a parameter set.
   The unit of measure appears in the table of the parameter set control in runtime.
3. Configure a start value in the "Start value" column for parameter set type items.
   The start value is used to pre-assign the corresponding parameters in a newly created parameter set in runtime.



Result

You have created a parameter set type with elements via a PLC user data type.

See also

Configuring user data types (Page 645)

"Parameter set types" editor (Page 856)

Changing a parameter set type with elements (Page 870)

Configuring the parameter set view (Page 880)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 875)

8.2.3 Changing a parameter set type with elements

Introduction

You have created one or more parameter set types with elements via a user data type. A parameter set type with elements determines the structure that is used for parameter sets on a machine. You have the following options to change parameter set types with elements:

- You can change one or more parameter set types automatically via a new version of the user data type.
- You can change a parameter set type manually via a new version of the user data type.
- You can change a parameter set type via another user data type.

Requirement

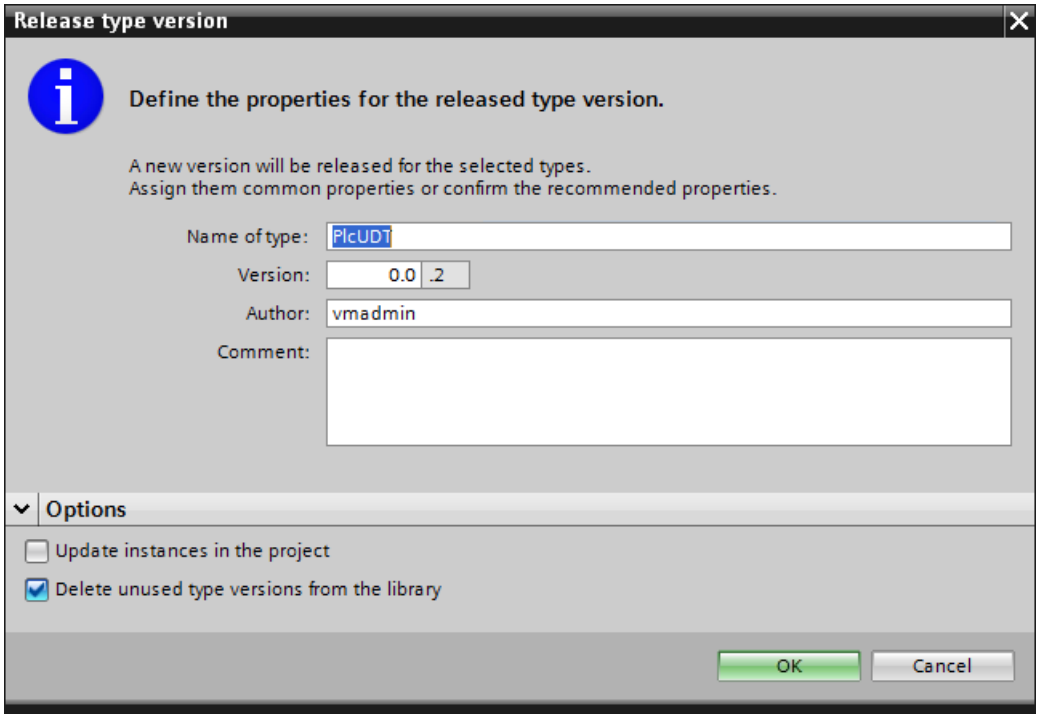
- One or more parameter set types with elements are created on the basis of an HMI or PLC user data type.

Changing parameter set types automatically via a new version of the user data type

To change parameter set types with elements automatically via a new version of the user data type, follow these steps:

1. Select the released user data type in the project library.
2. Select "Edit type" in the shortcut menu.
   In the case of an HMI user data type the "HMI user data types" editor is opened and a new user data type version with the "In process" status is generated in the project library. In the case of a PLC user data type the "Edit type" dialog is opened.
3. In the case of a PLC user data type click "OK" in the "Edit type" dialog.
   The "PLC data type" editor opens. A new user data type version with the "Testing" status is generated in the project library.
4. Change the user data type in the case of an HMI user data type in the "HMI user data types" editor and in the case of a PLC user data type in the "PLC data types" editor.

5. Select the new user data type version in the project library.
6. Select "Release version" in the shortcut menu.
   The "Release type version" dialog box opens.



7. If necessary, change the properties of the version:
   - Enter a unique name for the user data type in the "Type name" field.
   - Enter a valid version number for the version to be released in the "Version" field.
   - Under "Author" enter the editor of the version to be released.
   - Under "Comment" enter a comment on the version to be released.
8. If you want to revise the version management of the user data type, enable the "Delete unused type versions from the library".

9. To automatically change the parameter set type or types via the new user data type version, activate the option "Update instances in the project".
10. Click the "OK" button.
    The parameter set type or types are changed in the following way via the user data type version:
    - An element that you have not changed in the user data type, also remains in the parameter set type with all its settings.
    - An element that you have added new to the user data type is also added to the parameter set type.
    - The element that you have deleted from the user data type is also deleted from the parameter set type.
    - If you have changed the name of an element in the user data type, the parameter set type of the element's name is changed as well. All other properties of the element remain unchanged such as the start value or the display name.
    - If you have changed the data type of an element in the user data type, the parameter set type of the element's data type is changed as well. Other properties of the elements remain unchanged.

      Note

      An element whose data type you have changed is treated like a new element in runtime. Consequently the values of the element are deleted in the existing parameter sets and the start values are assigned as default to the element.

    - If you have changed a numerical data type of an element into a string data type in the user data type, the minimum and maximum values previously specified of the element are also removed.

Changing the parameter set type manually via a new version of the user data type

Proceed as follows to change a parameter set type with elements manually via a new version of the user data type:

1. Execute Steps 1 to 8 and 10 of the description above.
2. Open the "Parameter set types" folder in the project navigation.
3. Double-click a created parameter set type.
   The "Parameter set types" editor opens.
4. Select the new user data type version in the "Data type" column in the "Parameter set types" editor.
   The parameter set type is changed via the new user data type version as described in Step 10 of the above description.

Changing parameter set type via another user data type

Proceed as follows to change a parameter set type with elements manually via another user data type:

1. Open the "Parameter set types" folder in the project navigation.
2. Double-click a created parameter set type.
   The "Parameter set types" editor opens.
3. Select another released user data type in the "Data type" column in the "Parameter set types" editor.
   The structure of the parameter set type is created completely new in accordance with the structure of the other user data types.

See also

Creating a parameter set type with elements via an HMI user data type (Page 863)

Creating a parameter set type with elements via a PLC user data type (Page 866)

8.2.4 Assigning a tag of the data type HMI user data type to a parameter set type

Introduction

To exchange parameter sets between an HMI device and PLC in runtime, assign an external HMI tag to a parameter set type created via an HMI user data type in the Engineering System. To this purpose the HMI tag uses the HMI user data type as the data type with which you created the parameter set type.

Note

You can also assign an external HMI tag to a parameter set type that was created via a PLC user data type. In doing so the HMI tag uses the PLC user data type as the data type with which you created the parameter set type.

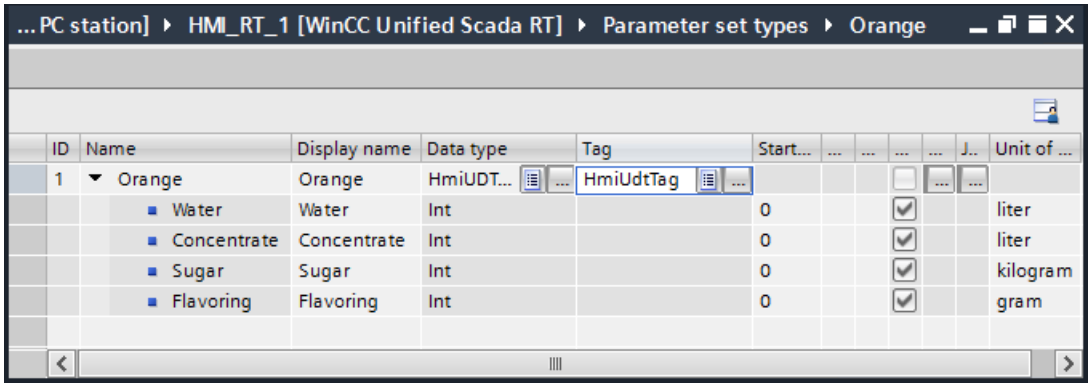
Requirement

- An HMI user data type is created in the HMI device.
- The SIMATIC S7-300/400 or SIMATIC S7 1200/1500 communication driver is set for the HMI user data type.
- User data type elements are added to the HMI user data type.
- The HMI user data type is released.
- A parameter set type with elements is created on the basis of the HMI user data type.
- An external HMI tag is created that uses the HMI user data type as the data type.

Procedure

To assign an external HMI tag of the data type HMI user data type to a parameter set type, proceed as follows:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click the created parameter set type.
   The "Parameter set types" editor opens. The Inspector window opens.
3. Select the created HMI tag of the data type "HMI user data type" in the column "Tag" in the "Parameter set types" editor.



Result

You have assigned an external HMI tag of the data type "HMI user data type" to a parameter set type.

See also

Creating a parameter set type with elements via an HMI user data type (Page 863)

Creating tags with a HMI user data type (Page 651)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 875)

Transferring and deleting parameter sets automatically (Page 876)

Transferring parameter sets (Page 897)

8.2.5 Assigning a tag of the data type "PLC user data type" to a parameter set type

Introduction

To exchange parameter sets between an HMI device and PLC in runtime, assign an external HMI tag to a parameter set type created via a PLC user data type in the Engineering System. In doing so the HMI tag uses the PLC user data type as the data type with which you created the parameter set type.

Note

You can also assign an external HMI tag to a parameter set type which was created via a HMI user data type. To this purpose the HMI tag uses the HMI user data type as the data type with which you created the parameter set type.

Requirement

- An HMI device has been created.
- A PLC user data type with user data type items is created in a SIMATIC S7-1200/1500 PLC.
- A parameter set type with elements is created on the basis of the PLC user data type.
- An external HMI tag is created that uses the PLC user data type as the data type.

Note

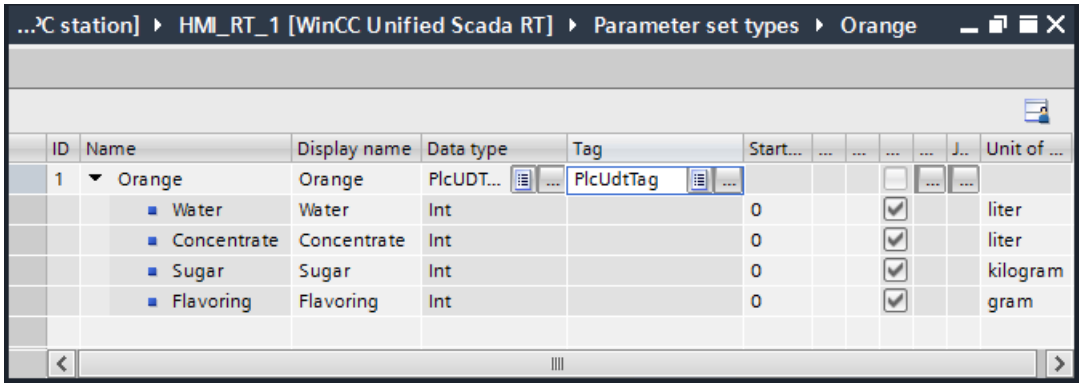
You have the following possibilities to create an external HMI tag of the data type "PLC user data type":

- Assign a PLC tag to an HMI tag, whereby the PLC tag is based on a PLC user data type.
- Assign a PLC data block that is based on a PLC user data type to an HMI tag.
- Assign a PLC data block element that is based on a PLC user data type to an HMI tag.

Procedure

To assign an external HMI tag of the data type "PLC user data type" to a parameter set, follow these steps:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click the created parameter set type.
   The "Parameter set types" editor opens. The Inspector window opens.
3. Select the created HMI tag of the data type "PLC user data type" in the "Tag" column in the "Parameter set types" editor.



Result

You have assigned an external HMI tag of the data type "PLC user data type" to a parameter set type.

See also

Creating a parameter set type with elements via a PLC user data type (Page 866)

External tags (Page 610)

Creating external tags (Page 626)

Assigning a tag of the data type HMI user data type to a parameter set type (Page 873)

Transferring and deleting parameter sets automatically (Page 876)

Transferring parameter sets (Page 897)

8.2.6 Transferring and deleting parameter sets automatically

Introduction

In the Engineering System you assign control tags to a parameter set type with elements. The control tags serve to automatically transfer or delete parameter sets between an HMI device and PLC in runtime. In the process either the control program or the HMI device controls the automatic transfer or deleting via control requests.

Requirement

- A parameter set type with elements is created on the basis of an HMI or PLC user data type.
- An external HMI tag that uses the HMI or PLC user data type as the data type is assigned to the parameter set type.
- Two control tags with numerical data type are created, for example, "ParameterSetIDTag" and "JobIDTag".

Note

If you want to automatically transfer or delete parameter sets by means of the control program, create 2 external control tags. If you want to automatically transfer or delete parameter sets by means of the HMI device, however, create 2 internal control tags.

Note

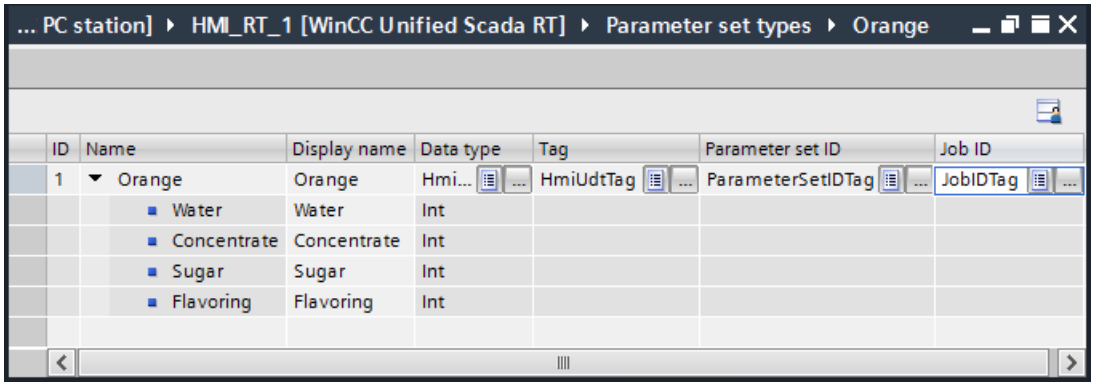
The control tags must use a signed data type.

- The "Toolbox" task card is open.

Assigning control tags to a parameter set type with elements

Proceed as follows to assign control tags to a parameter set type with elements in the Engineering System:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click the created parameter set type.
   The "Parameter set types" editor opens. The Inspector window opens.
3. Select the control tag "ParameterSetIDTag" in the "Parameter set ID" column in the "Parameter set types" editor.
4. Select the "JobIDTag" control tag in the "Job ID" column.



Note

Please observe each of the following rules to not obtain any errors when compiling the project:

- If you set a control tag in the "Parameter set ID" column of the "Parameter set types" editor, you also set a control tag in the "Job ID" column.
- Do not set the same control tag in the columns "Parameter set ID" and "Job ID" in the "Parameter set types" editor.
- If you assign a control tag to a parameter set type in the "Parameter set types" editor, do not assign the same control tag to another parameter set type.

Reading a parameter set from the PLC

Proceed as follows to automatically read a parameter set from the PLC and store it in the parameter set memory in runtime:

1. Automatically set the control tag "ParameterSetIDTag" to an ID of an existing parameter set.
2. Automatically set the control tag "JobIDTag" to the control job ID "6".
   The control job is executed. In the case of success the control tag "JobIDTag" is set to the value "0", otherwise to the value "-1".

Note

If you set a non-existing parameter set ID and the control job ID to "6", a new parameter set is created with the parameter set values available in the PLC.

Writing a parameter set to the PLC

To automatically load a parameter set from the parameter set memory in runtime and write it into the PLC, follow these steps:

1. Automatically set the control tag "ParameterSetIDTag" to an ID of an existing parameter set.
2. Set the control tag "JobIDTag" automatically to the control job ID "7".
   The control job is executed. In the case of success the control tag "JobIDTag" is set to the value "0", otherwise to the value "-1".

Deleting a parameter set

To automatically delete a parameter set from the parameter set memory in runtime, follow these steps:

1. Automatically set the control tag "ParameterSetIDTag" to an ID of an existing parameter set.
2. Set the control tag "JobIDTag" automatically to the control job ID "8".
   The control job is executed. In the case of success the control tag "JobIDTag" is set to the value "0", otherwise to the value "-1".

Result

You have assigned control tags to a parameter set type with items in the engineering system and automatically transferred and deleted parameter sets between the HMI device and PLC via the control tags in runtime.

See also

Assigning a tag of the data type HMI user data type to a parameter set type (Page 873)

Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 875)

External tags (Page 610)

Creating external tags (Page 626)

Configuring the parameter set view (Page 880)

Transferring parameter sets via scripts (Page 879)

Managing parameter sets (Page 885)

Transferring parameter sets (Page 897)

8.2.7 Transferring parameter sets via scripts

Introduction

In runtime, you can transfer parameter sets via local scripts between the HMI device and PLC. To do so configure the local scripts in the Engineering Systems at events of screen objects or at the "Update" event of tasks.

Note

You can also use system functions in the function list or in scripts to transfer parameter sets between HMI device and PLC:

- With the system function "ReadAndSaveParameterSet" or "ReadAndSaveParameterSet", a parameter set is read from the PLC and saved in the parameter set memory.
- With the system function "LoadAndWriteParameterSet" or "LoadAndWriteParameterSet", a parameter set is loaded from the parameter set memory and written to the PLC.

Code examples for transferring parameter sets

To load a parameter set from the parameter set memory and write it into the PLC, use the following code example:

```
let ps1 = HMIRuntime.ParameterSetTypes('MyPST1').ParameterSets(1);
ps1.LoadAndWrite(true);
```

To read a parameter set from the PLC and store it in the parameter set memory, use the following code example:

```
let ps1 = HMIRuntime.ParameterSetTypes('MyPST1').ParameterSets(1);
ps1.ReadAndSave(HMIRuntime.ParameterSetTypes.Enums.hmiOverwrite.Disabled, true);
```

	**Tip for an efficient procedure**
You can find the code examples in the shortcut menu of the "Scripts" editor under "Snippets > HMI Runtime > Parameter Set".	

See also

Local scripts (Page 980)

Transferring and deleting parameter sets automatically (Page 876)

Transferring parameter sets (Page 897)

LoadAndWriteParameterSet (Page 924)

ReadAndSaveParameterSet (Page 932)

8.2.8 Configuring the parameter set view

Introduction

To display, manage and exchange parameter sets with the PLC in runtime, use a parameter set control. You configure a parameter set control in the engineering system.

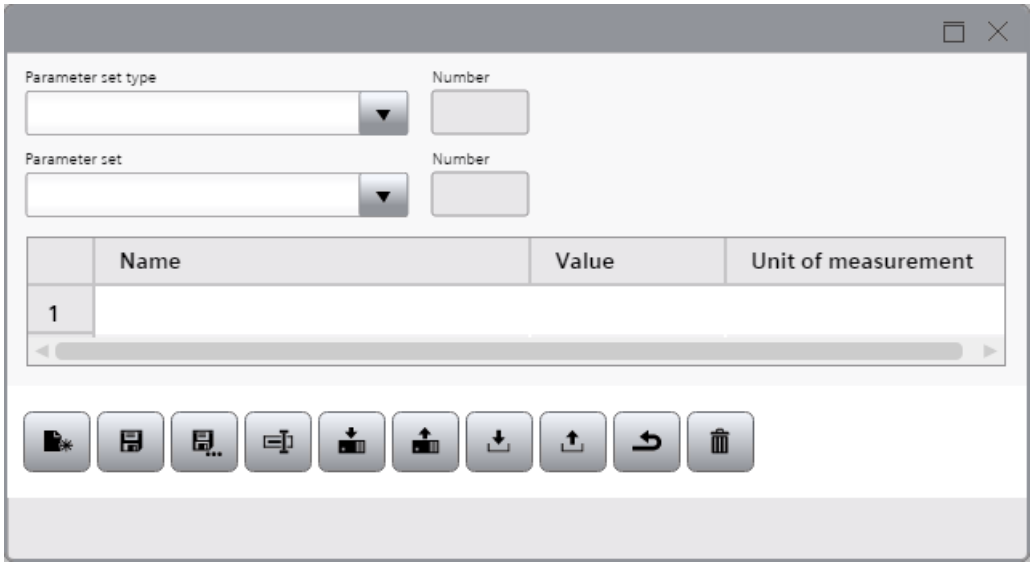
Requirement

- At least one parameter set type with elements has been created.
- A screen is open.
- The "Toolbox" task card is open.

Procedure

To configure a parameter set control, proceed as follows:

1. Insert the "Parameter set control" object from the "Tools" task card into the screen.



2. Set the desired height, width and position for the parameter set control in the Inspector window under "Properties > Size and Position".
3. If you only want to use a particular parameter set type with its parameter sets in runtime, select the desired parameter set type under "Properties > General > Fixed parameter set type".

   Note

   The parameter set type is only visible in the parameter set control in runtime when the extended style is used in the runtime settings under "General > Screen".

   Note

   A tag that you use as a dynamization tag of the fixed parameter set type in the parameter set control must have the data type "String" or "WString".

4. Change the labels of the fields, if required.
5. If required, change the display of the parameter table under "Properties > Miscellaneous > Parameter view".

   Note

   To change the visibility of columns, click on "Columns" under "Properties > Miscellaneous > Parameter view". In the extended parameter control display, you can enable or disable the visibility of individual columns under "Visibility". Changing the visibility via the preset value in the "Blocks" property is not supported.

6. In "Properties > Miscellaneous > Editing mode", configure the activation status of the toolbar buttons "Create", "Save", "Save as", "Rename" and "Delete" as required.
   These toolbar buttons are used to edit parameter sets.

7. If you want to hide specific buttons in the toolbar, deactivate the "Visibility" property under "Properties > Miscellaneous > Toolbar > Elements" in the settings of the corresponding button.
8. Under "Properties > Miscellaneous > Time zone", change the time zone as required by entering a different numerical value.
   The numerical value stands for a time zone, for example:
   - "-1" stands for UTC-1h (Central European Time, standard time)
   - "1" stands for UTC-12h (International Date Line West)
   - "2" stands for UTC-11h (Hawaii)

See also

Configuring reordering of the columns (Page 392)

Parameter set control (Page 860)

Creating a parameter set type with elements via an HMI user data type (Page 863)

Creating a parameter set type with elements via a PLC user data type (Page 866)

Transferring and deleting parameter sets automatically (Page 876)

Managing parameter sets (Page 885)

Exporting and importing parameter sets (Page 891)

Transferring parameter sets (Page 897)

8.2.9 Assigning an edit tag to a parameter set item

Introduction

To adjust the values of the parameter sets in the HMI device and the PLC in runtime, you can assign a local session HMI tag to a parameter set type in the Engineering System.

The edit tag enables you to manage the values of a parameter set in a screen. To do this, you use system functions or scripts to transfer values between the HMI device and the PLC.

Requirement

- An HMI device has been created.
- An HMI or PLC user data type with user data type items is created in a SIMATIC S7-1200/1500 PLC.
- A parameter set type with parameter set items based on the HMI or PLC user data type is created.
- A local session HMI tag with the same number of items and the same data type is created. The HMI tag must not contain any information about the range (Upper 2, Lower 2).

Procedure

	**Tip for an efficient procedure**
To create the same structure for the edit tag as for the parameter set tag, copy the corresponding data type in the library. Change the connection to "Internal connection" in the copy. In this way, you make the data type available for a local session edit tag.	

To assign a local session HMI tag to a parameter set, follow these steps:

1. Open the "Parameter set types" folder in the project tree.
2. Double-click the created parameter set type.
   The "Parameter set types" editor opens.
3. In the "Parameter set types" editor, select a local session HMI tag in the "Edit tag" column.

Result

You have assigned a local session HMI tag to a parameter set type.

8.2.10 Configuring parameter sets without parameter set control

Introduction

In one or more screens, you configure an individual input mask as an extension or replacement of the parameter set control. You create the input mask from I/O fields and other display and operating objects. System functions or scripts are used to configure the parameter set functionality, such as saving parameter sets.

Use

Configuring directly in screens allows you to display and manage parameter sets according to your needs. You can spread parameter sets containing lots of entries across several screens to create a better overview. For example, you can configure a separate screen with the corresponding input masks for the parameter sets for each area of the plant.

You can visually map your machine in a screen using graphical screen objects. This enables you to display parameter settings clearly by positioning I/O fields directly next to machine elements such as axes or guide rails. This is how you produce a direct relationship between the values and the machine.

You configure system functions and scripts with which you call up system information, create and manage parameter sets or adapt active parameter sets via edit tags.

Requirement

- The parameter set type is created.
- The "Screens" editor is open.

Procedure

To configure a screen with which you can display and manage parameter sets, follow these steps:

1. Configure a screen and create the I/O fields for the recipe input mask in it.
   You can create multiple screens to suit the size and complexity of the recipe.
2. Configure the I/O fields with the tags that you have connected to the parameter set items.
3. Configure I/O fields for the parameter set type and parameter set selection.
4. Configure the system functions for editing parameter sets on the configured operator controls.
   Operator controls are configured buttons in the screen or function keys on the HMI device. In the "Scripts" editor, you will find snippets with the system functions under "HMI Runtime > Parameter set".

Alternative procedure

1. Configure a parameter set control as a selection list for parameter set types and parameter sets.
2. Hide the buttons that are not required in the parameter set control.
3. Configure the system functions for editing parameter sets on the configured operator controls.

Result

A screen that offers the possibility to display and manage parameter sets has been created.

With the use of scripts and the use of system functions, you have the following possibilities:

- Reading the name of a parameter set
- Reading the name of a type of parameter set
- Reading a selected parameter set
- Writing a parameter set to the PLC
- Trigger event on parameter set
- Display of system information of the status display in a separate screen

See also

System functions (Page 909)

ParameterSetTypes (Page 1233)

ParameterSetType (Page 1234)

8.3 Using parameter sets in runtime

8.3.1 Managing parameter sets

Introduction

You manage parameter sets for different productions in a parameter set control in runtime. You have the following options for managing parameter sets:

- Create new parameter sets
- Copy parameter sets
- Change parameter sets
- Delete parameter sets
- Rename parameter sets

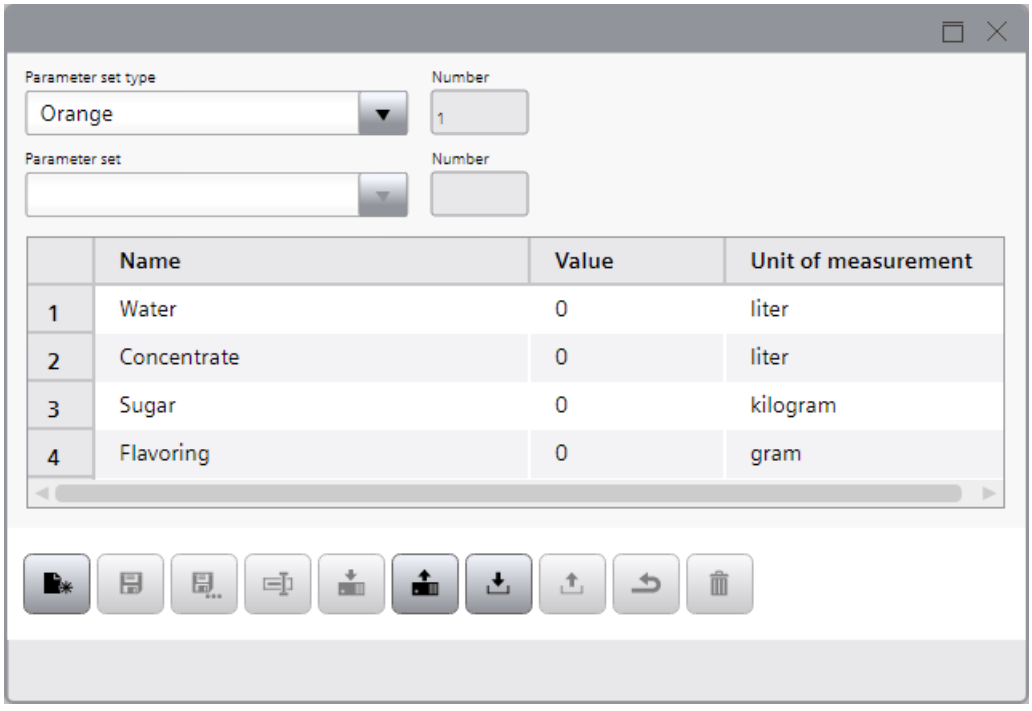
Requirement

- At least one parameter set type with elements has been created.
- A parameter set control has been configured.
- The project is in runtime.

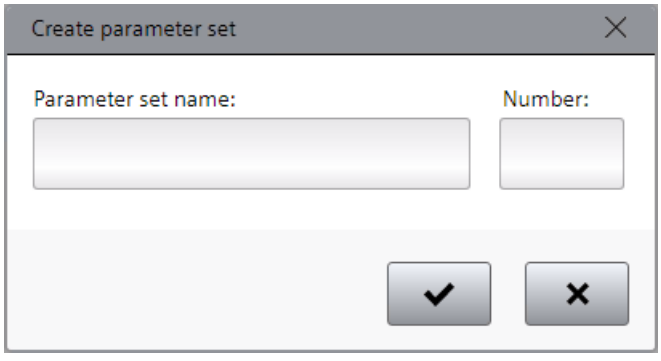
Creating a new parameter set

To create a new parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type for which you want to create a new parameter set.
   The elements of the selected parameter set type are displayed in the table.

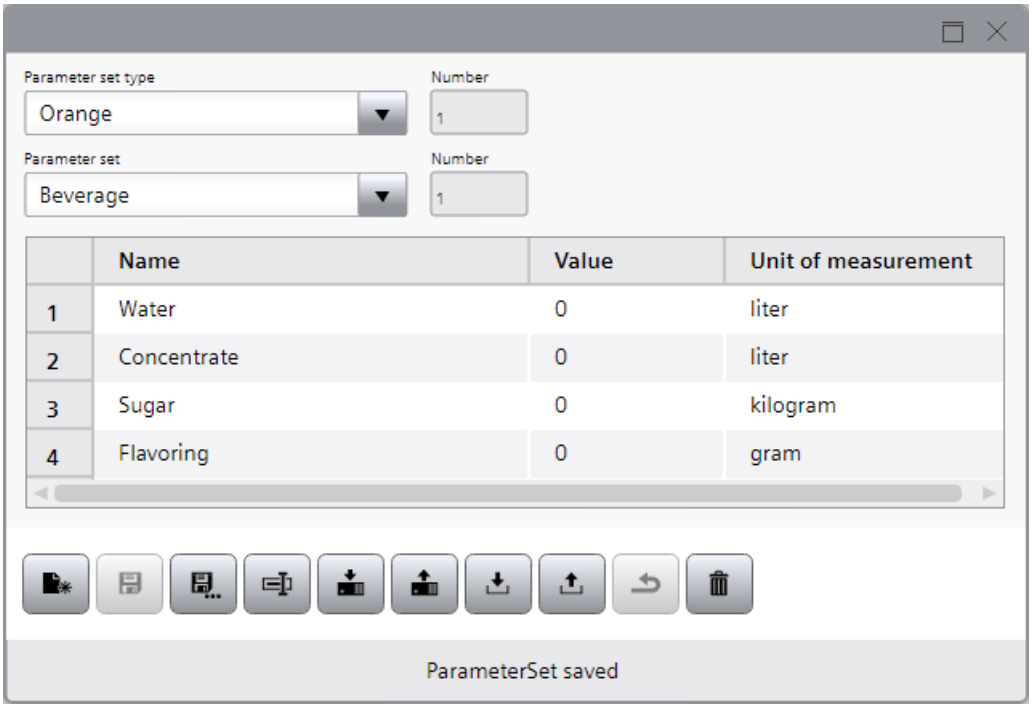


2. Click the "Create" button.
   The "Create parameter set" dialog opens.



3. Enter a unique parameter set name under "Parameter set name".
4. Enter a unique parameter set ID under "Number".

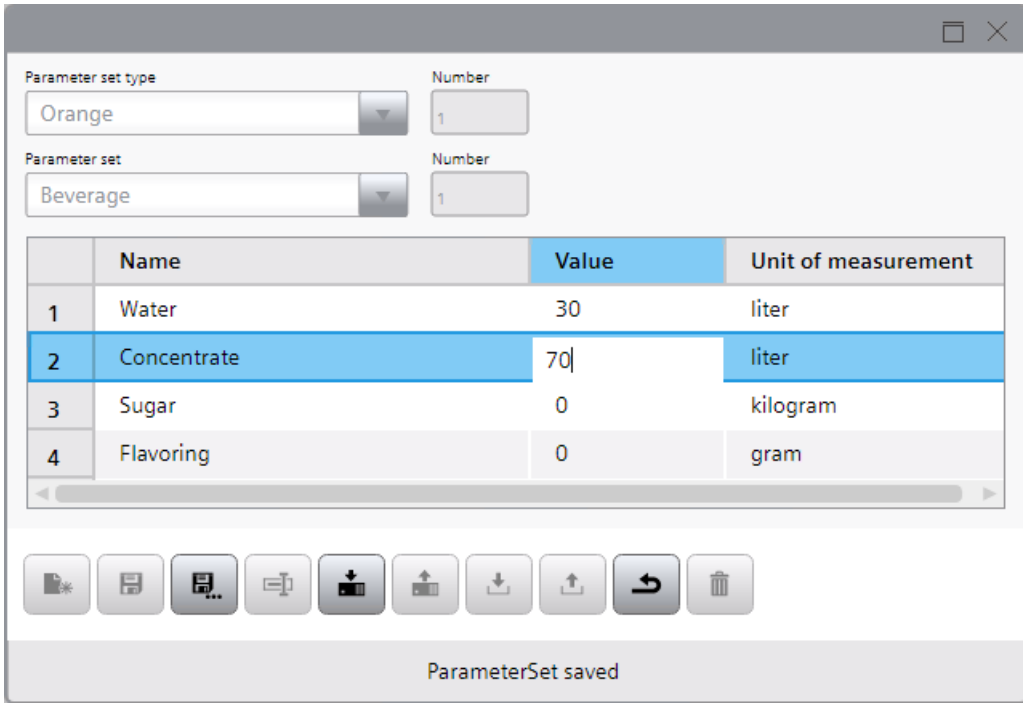
5. Confirm the dialog.
   A new parameter set has been created and saved. The parameters of the new parameter set are displayed in the table. The parameters have the same values in the columns "Name" and "Unit of measurement" as the elements of the previously selected parameter set type. The defined start values are applied for the "Value" column. If you have not defined start values, the corresponding default values are used.



Note

If you do not make any entries in the "Create parameter set" dialog and confirm the dialog, a new parameter set is also created and saved. In this case the new parameter set, however, has a unique parameter set name and a unique parameter set ID which were both automatically assigned by the system.

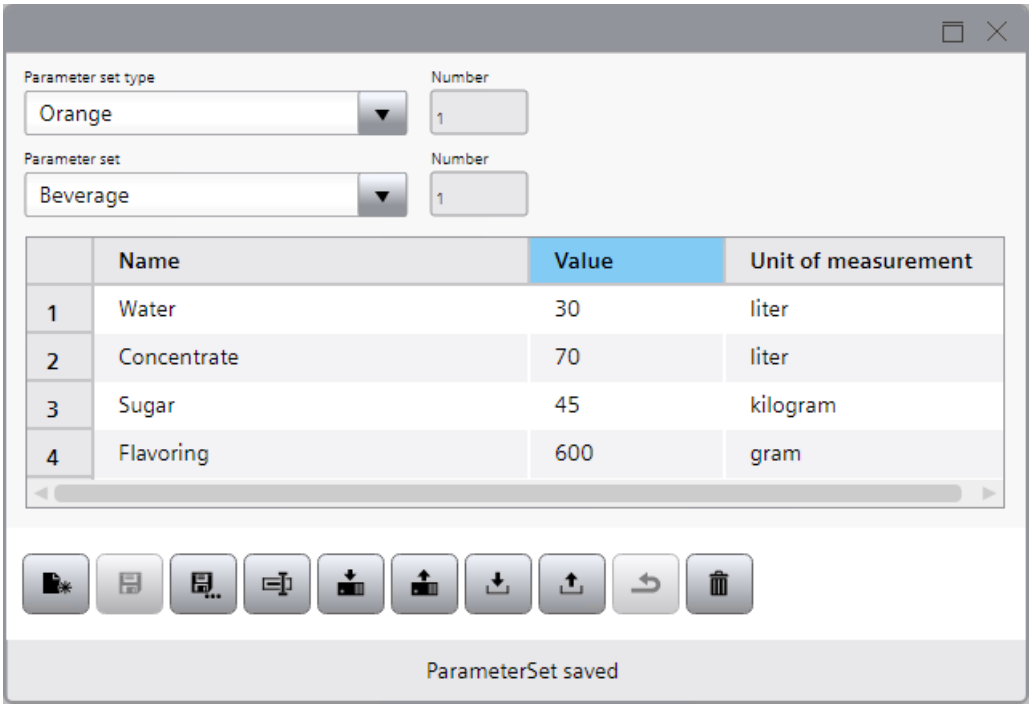
6. Enter values for the parameters in the "Value" column.
   Depending on the configuration, the parameters already contain start values.



Note

The character ' is not permitted in the value of a parameter set.

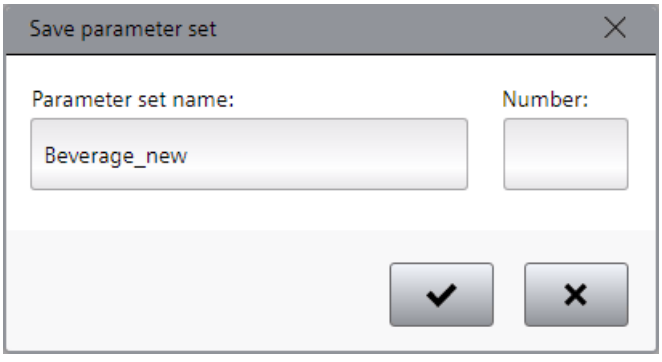
7. Click the "Save" button.



Copying a parameter set

To copy a parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type in which you want to copy an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to copy.
   The parameters of the selected parameter set are displayed in the table.
3. Click the "Save as" button.
   The "Save parameter set" dialog opens. A unique parameter set name is pre-assigned to the "Parameter set name" field.



4. Enter a different unique parameter set name under "Parameter set name" as required.

5. Enter a unique parameter set ID under "Number" as required.
6. Confirm the dialog.

Note

If you do not enter a parameter set ID in the "Save parameter set" dialog and confirm the dialog, a unique parameter set ID is automatically assigned to the new parameter set.

Changing the parameter set

To change a parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type in which you want to change an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to change.
   The parameters of the selected parameter set are displayed in the table.
3. Edit the values of the parameters in the "Value" column.
4. Click the "Save" button.

Deleting a parameter set

To delete a parameter set, proceed as follows:

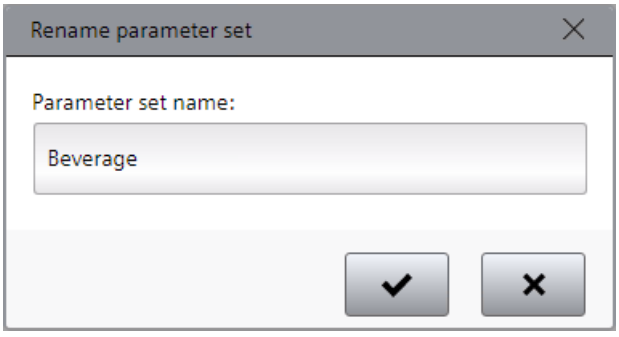
1. In the "Parameter set type" field select the parameter set type in which you want to delete an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to delete.
   The parameters of the selected parameter set are displayed in the table.
3. Click "Delete".

Renaming a parameter set

To rename a parameter set, proceed as follows:

1. In the "Parameter set type" field, select the parameter set type in which you want to rename an existing parameter set.
   The elements of the selected parameter set type are displayed in the table.
2. In the "Parameter set" field, select the parameter set you want to rename.
   The parameters of the selected parameter set are displayed in the table.

3. Click the "Rename" button.
   The "Rename parameter set" dialog opens.



4. Enter a different unique name for the parameter set under "Parameter set name".
5. Confirm the dialog.

See also

Parameter set control (Page 860)

Configuring the parameter set view (Page 880)

Transferring and deleting parameter sets automatically (Page 876)

Exporting and importing parameter sets (Page 891)

Transferring parameter sets (Page 897)

8.3.2 Exporting and importing parameter sets

Introduction

In a parameter set control in runtime you export parameter sets from the parameter set memory to a "*.tsv" file to be able to edit them a text editor. In a parameter set control in runtime you furthermore import parameter sets from a "*.tsv" file into the parameter set memory. A "*.tsv" file is a text file that uses the tabulator as a list separator.

Note

To export and import the parameter sets, you can also use the system functions in the function list or in the scripts:

- With the system function "ExportParameterSets" or "ExportParameterSets", the parameter sets are exported from the parameter set memory to a "*.tsv" file.
- With the system function "ImportParameterSets" or "ImportParameterSets", the parameter records are imported from a "*.tsv" file into the parameter set memory.

If the parameter `OutputStatus` is set to `True`, a status message is output in an alarm control configured in the screen.

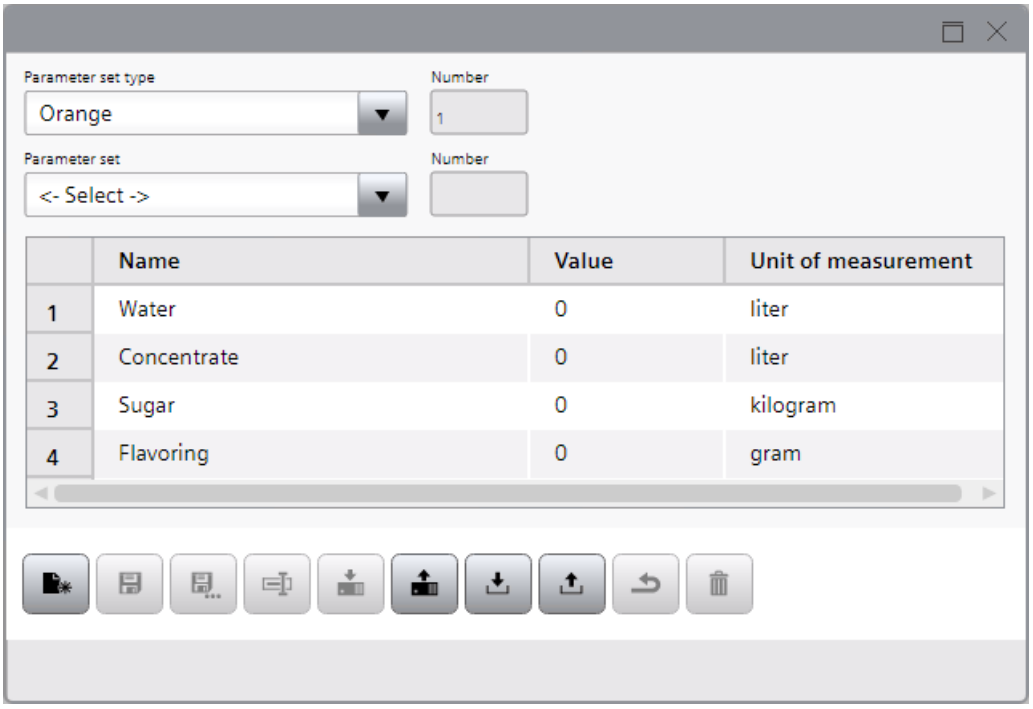
Requirement

- At least one parameter set type with elements has been created.
- A parameter set control has been configured.
- The project is in runtime.

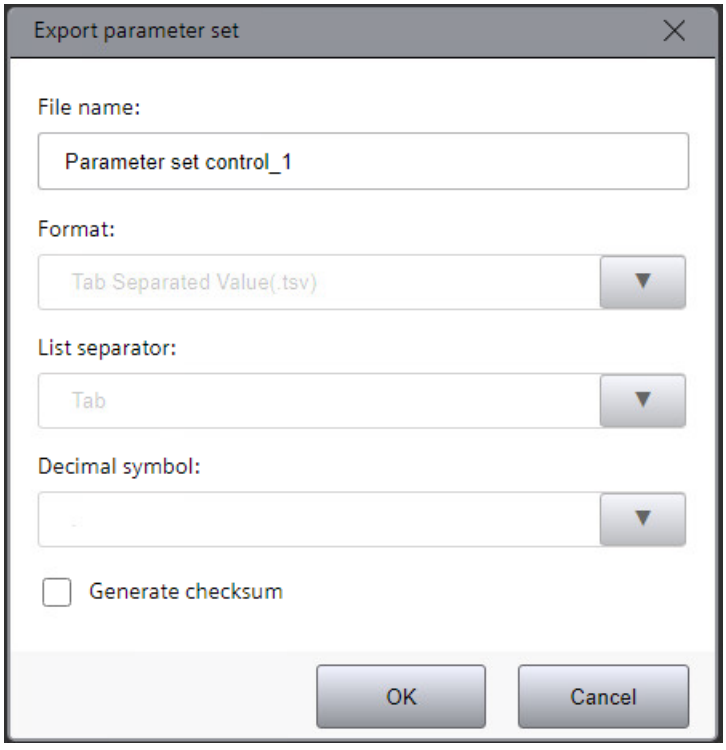
Exporting parameter sets of a parameter set type

Follow these steps to export the parameter sets of a parameter set type:

1. In the "Parameter set type" field, select the parameter set type whose parameter sets you want to export.

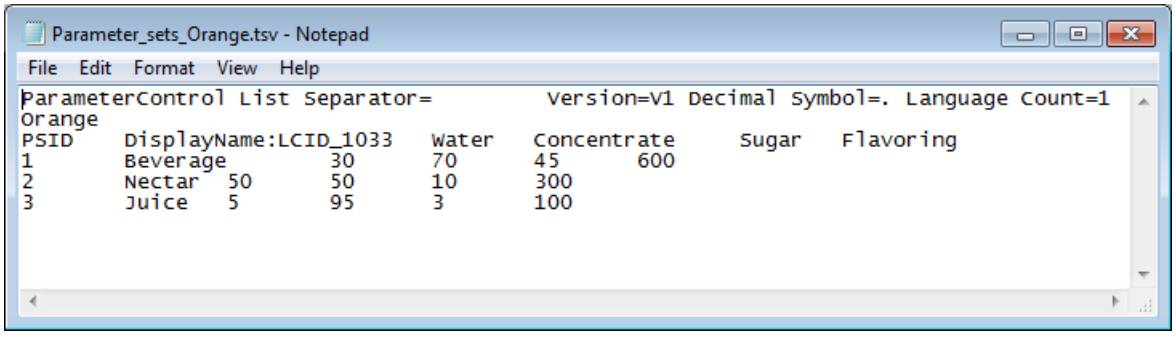


2. Click the "Export" button.
The "Export parameter set" dialog box opens. The name of the parameter set control is pre-assigned in the "File name" field.



3. If appropriate, change the name of the file to which you want to export the parameter sets under "File name".

4. Enable "Generate checksum" to export the parameter data set with a checksum.
   Parameter data sets with a checksum cannot be imported if they have been manipulated in the meantime.
5. Confirm the dialog.
   The parameter sets are exported to a ".tsv" file.
   This file is stored according to the download settings.
   The file has the following structure:
   - The first line contains the file header. The file header consists of identifier, delimiter, version of the exported file, decimal symbol and information on the number of languages in which the name of the parameter sets is stored.
     The line must not be changed. Otherwise, it is not possible to import parameter sets.
   - The second line contains the name of the parameter set type.
   - The third row contains the headers for parameter sets. LCID of the language and the names of the parameter set type items are listed.
     The header for parameter sets must not be changed.
   - From the fourth line on the parameter sets are listed.



Edit exported file

1. You can customize the file to meet your needs:
   - Change the values of existing parameters.
   - Add parameter sets.

   To be able to import the parameter sets after editing, note the following information:

   Note
   - The parameters must be valid for the defined data type.
   - The parameters must be within the limits defined in the parameter set type item.
   - ID and name of the individual parameter sets must be unique.

2. Save the changes.

Importing parameter sets into a parameter set type

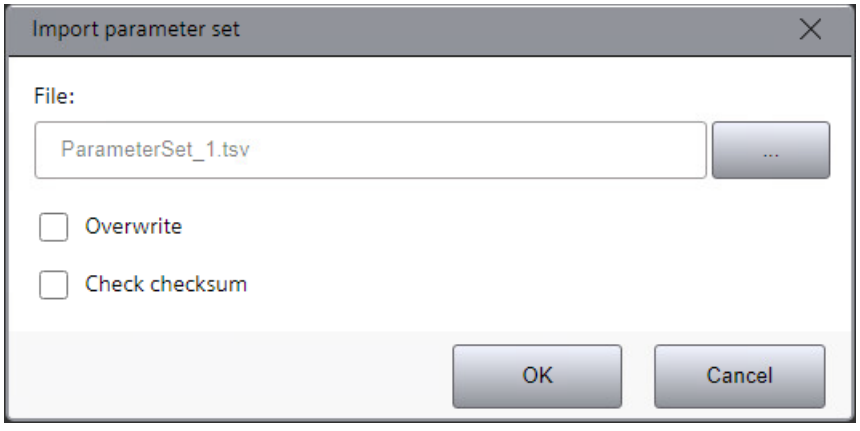
To be able to import parameter sets, note the following requirements:

Note

- The import file must have the same file header and the same header for parameter sets as the export file. Otherwise, it is not possible to import parameter sets.
- There is no parameter set with the same display name in any of the configured languages.
- The numerical values in the import file are within the permitted value range of the corresponding configured data type.

To import parameter sets into a parameter set type, follow these steps:

1. In the "Parameter set type" field, select the parameter set type into which you want to import the parameter sets.
2. Click the "Import" button.
   The "Import parameter set" dialog box opens.



3. Select the file from which you want to import the parameter sets.
4. To overwrite parameter sets in the parameter set control that have the same ID as the imported parameter sets, activate the "Overwrite" option.

   Note

   If you deactivate overwriting and if a parameter set with the same ID or the same parameter set name exists in the parameter set control, the import of parameter sets is not possible.

   Any added parameter sets whose IDs and parameter set names deviate from the existing parameter sets are imported regardless of the "Overwrite" option.
5. Enable "Check checksum" when importing a parameter data set exported with the "Generate checksum" option.
6. Confirm the dialog.
   The parameter sets are imported to the parameter set type.

See also

SysFct.ExportParameterSets() (Page 1248)

SysFct.ImportParameterSets() (Page 1254)

Parameter set control (Page 860)

Configuring the parameter set view (Page 880)

Managing parameter sets (Page 885)

8.3.3 Transferring parameter sets

Introduction

You have assigned an external HMI tag of the data type HMI or PLC user data type to a parameter set type. In a parameter set control in runtime you transfer the values of parameter sets to the PLC via the HMI tag. The parameter set values are used to set up machines for different productions.

In a parameter set control in runtime you furthermore read active parameter sets from the PLC into the parameter set control via the HMI tag. The read parameter set values are stored in the parameter set memory. By reading from the PLC, you call up currently used values from production machines for later use.

Note

You can also use system functions in the function list or in scripts to transfer parameter sets between HMI device and PLC:

- With the system function "ReadAndSaveParameterSet" or "ReadAndSaveParameterSet", a parameter set is read from the PLC and saved in the parameter set memory.
- With the system function "LoadAndWriteParameterSet" or "LoadAndWriteParameterSet", a parameter set is loaded from the parameter set memory and written to the PLC.

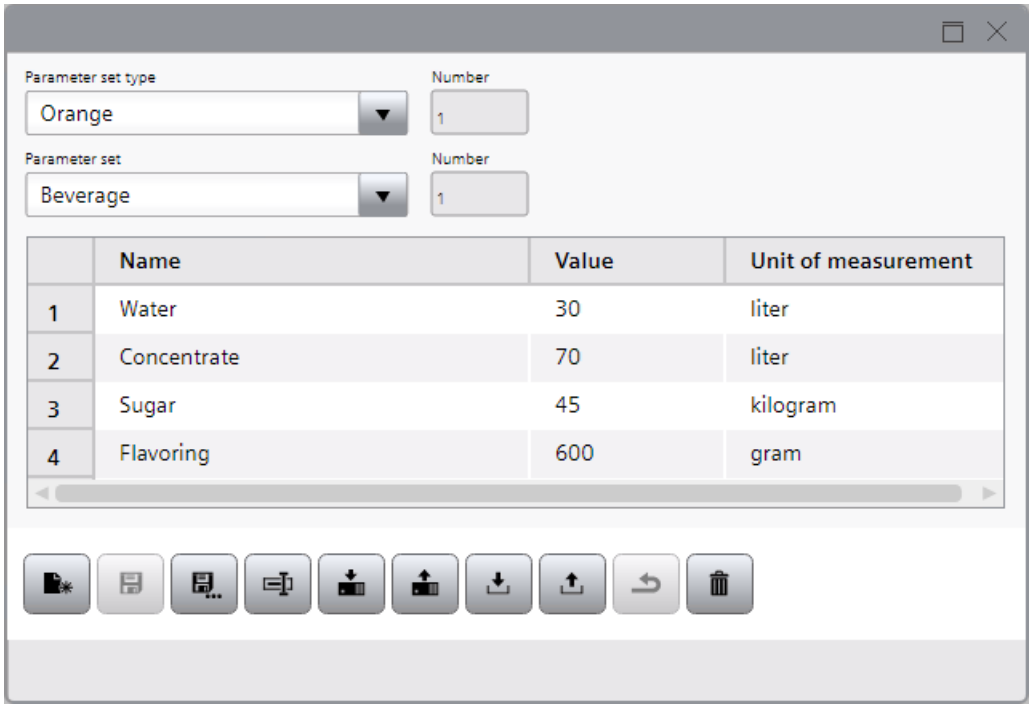
Requirement

- A parameter set type with elements has been created.
- An external HMI tag of the data type HMI or PLC user data type is assigned to the parameter set type.
- A parameter set control has been configured.
- The project is in runtime.
- At least one parameter set has been created in the parameter set type.

Transferring a parameter set to the PLC

To transfer a parameter set to the PLC, follow these steps:

1. In the "Parameter set type" field, select the parameter set type.
2. In the "Parameter set" field, select the parameter set whose values you want to transfer to the PLC.



3. Click the "Write to PLC" button.

Reading a parameter set from PLC

To read a parameter set from the PLC, follow these steps:

1. In the "Parameter set type" field, select the parameter set type.
2. In the "Parameter set" field, select the parameter set whose values you want to read from the PLC.

Note

If do you not select a parameter set in the in the "Parameter set" field, a new parameter set is created in the parameter set control while reading from the PLC.

3. Click the "Read from PLC" button.

Note

A parameter set cannot be read from the PLC if minimum and/or maximum values are defined for a parameter set type item and the value in the parameter set to be transferred is outside this range. An alarm is triggered.

Result

You have transferred the values of parameter sets between the HMI device and PLC.

See also

LoadAndWriteParameterSet (Page 924)
ReadAndSaveParameterSet (Page 932)
Assigning a tag of the data type HMI user data type to a parameter set type (Page 873)
Assigning a tag of the data type "PLC user data type" to a parameter set type (Page 875)
Parameter set control (Page 860)
Configuring the parameter set view (Page 880)
Managing parameter sets (Page 885)
Transferring and deleting parameter sets automatically (Page 876)
Transferring parameter sets via scripts (Page 879)

Using system functions

9

9.1 Working with function lists

9.1.1 Basics of the function list

Introduction

A function list performs one or more functions when the configured event occurs.

The following are available:

- System functions
- Functions which you configure in global modules
- Functions which you configure in script modules

Principle

The function list is configured for an event of the following objects:

- Screen
- Screen object
- Task

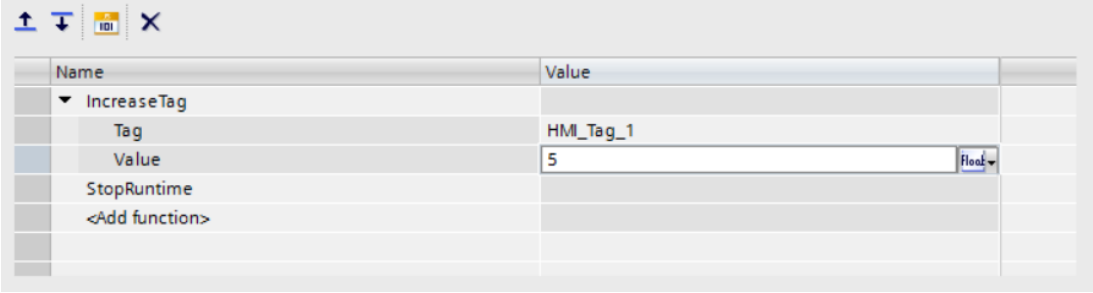
You can configure exactly one function list for each event. Which events are available depends on the selected object. Events occur only when the project is in runtime.

Events include:

- Execution of a task
- Pressing of a button

The function list is opened in the Inspector window under "Properties > Events" as soon as an object has been selected.

Layout



The function list is divided into two columns. In the "Name" column you see the names of the functions and the corresponding parameters.

In the Values column you assign values to the parameters. You use different parameter types depending on the parameter.

The following buttons are located above the function list:

Button	Function
	Move function up
	Move function down
	Display parameters
	Hide parameter
	Convert function list to script
	Delete function list

If the function list cannot be edited, the buttons are locked. This is the case, for example, with reference projects.

See also

Editing a function list (Page 906)

System functions (Page 909)

"Scripts" editor (Page 981)

9.1.2 Input support

The function list supports you in:

- Function input
- Input of parameter values
- Troubleshooting

Function input

You have several options for entering a function in the function list:

- Enter the complete or partial name of the function.
  Autocomplete is supported.
- Select the "<Add function>" field and open the selection menu.
  The available functions are sorted by category.
- Select the "<Add function>" field and select the list icon.
  The available functions are displayed in alphabetical order.

The system functions provided in the function list differ depending on the object selected. For example, system functions of the "Screen" category are only available for screens and screen items.

Input of parameter values

The parameters required in functions are displayed in the function list and differ depending on the function selected.

You assign a value to each parameter.

The parameter type determines which values the parameter can accept.

The parameter type is either fixed by default or you can choose between several types.

Which parameter types are available depends on the respective parameter.

The following parameter types are implemented in the function list:

- Integer
- UInteger
- Double
- Bool
- String
- Color
- HMI tag: Specify a configured HMI tag. Autocomplete is supported.
- Screen object: Specify a configured screen item in the current screen. Autocomplete is supported.
- Selection: Sets the current screen as value. If this type is selected, the value cannot be edited.
- Text list
- Screen window
- Runtime language

Note

HMI tags and screen items can be renamed without updating the function list. Objects of the WinCC Unified object model are referenced in the function list.

Optional parameters are marked with "(optional)".

Parameters of functions that you have configured in global modules are always optional.

Assigning parameters via drag-and-drop operation

For some objects you can assign the objects to parameters of the function list using drag-and-drop operation.

You assign the following objects from the detail view to function list parameters using the drag-and-drop feature:

- HMI tag
- PLC tag
  If you assign a PLC tag by drag-and-drop operation, a connected HMI tag is created automatically. The HMI tag is linked to the parameter.
- Screen
  In addition, you can assign screens from the project tree to parameters in the function list using the drag-and drop features.
- Text list

Troubleshooting

The project data is tested in the background during the configuration.

To inform you of errors, missing or incorrect entries in the function list are highlighted in red:

- At least one function is not completely supplied with parameters.
- At least one function is contained which is not supported by the selected HMI device, for example, by changing the device type.

Note

Missing screen items and HMI tags can be created later. Objects of the WinCC Unified object model are referenced in the function list.

Alarms during the compiling and loading of a project are displayed in the Inspector window in the "Info > Compile" tab.

The function list supports you by displaying missing or incorrect entries directly for editing:

- To go directly to the function list, select the green arrow  .

See also

"Scripts" editor (Page 981)

System functions (Page 909)

9.1.3 Configuring a function list

Restrictions for events

- "Activated" and "Deactivated" events:
  If the focus is on the affected screen item, scripts are executed at the "activated" and "deactivated" events.
- "Press key" and "Release key" events:
  The events "Press key" and "Release key" can only be queried when a keyboard is connected.

Requirement

One of the following objects is configured:

- Screen
- Screen item
- Task

Procedure

1. Select the object.
2. Go to "Properties > Events" in the Inspector window.
   The function list opens.
3. Select an event.
4. Select "<Add function>" in the function list.
5. Enter the function.
   If the desired function has parameters, the parameters are displayed.
6. Define the parameter values.
7. To add more functions, repeat steps 4.) to 6.).
8. Save the project.

Result

- The function list is configured.
- In addition, to the configured event, the status of the function list is displayed in the Inspector window.
- The function list is executed from top to bottom when the configured event occurs in runtime.

9.1.4 Editing a function list

Editing function lists

The function list is executed from top to bottom. You have the following options for editing function lists:

	"Move function up"
	"Move function down"
	"Convert function list to script"
	"Delete function list"

Requirement

- The function list is open.
- At least one function is configured at an event.

Changing the order of functions

You move a function within the function list as follows:

1. Select the name of the function or an associated parameter.
2. Select the "Move function up" or "Move function down" button.
   If the function is already at the first or last position in the list, pressing the button has no effect.

Replacing a function

1. To replace a function, enter the name of another function in the input field. All parameter settings of the replaced function are deleted.

Converting a function list to a local script

1. To convert the function list to a local script, use the "Convert function list to script" button.
2. The "Scripts" editor opens. Adapt the script to your requirements.

The function list can be converted to a local script even if the parameter specifications are incorrect or incomplete. The parameter specifications must be adjusted accordingly in the script.

Note

This action can only be revoked using the  "Undo" button.

Delete function list

1. To delete the entire function list of an event, select the "Delete function list" button.

Deleting functions

To delete a single function, follow these steps:

1. Select the name of the function.
2. Press <Del>.

See also

"Scripts" editor (Page 981)

9.1.5 Using a screen item to specify the value of a parameter

Introduction

For some parameters, you can assign "Screen item" as parameter type.

For example, enter the "Value" parameter of the "IncreaseTag" function via the I/O field in runtime.

Note

The assigned screen item must have the property "Process value".

Possible screen items are, for example:

- I/O field
- Bar
- Slider
- Radio button

The value entered in the screen item must correspond to the data type expected by the system function.

If the data types do not match, convert the value using an additional tag.

Requirement

- A screen is configured.
- A suitable screen object (e.g. I/O field) is configured.
- A suitable system function (e.g. "IncreaseTag") is created in the function list.

Procedure

To assign a parameter to a screen item and convert the value entered in the screen item, proceed as follows:

1. Assign the desired screen item to the parameter.
2. Select the screen item.
3. Go to "Properties > General > Process value" in the Inspector window.
4. Select "Tag" in the "Dynamization" column.
   The tag selection range is displayed.
5. Select the selection button ....
6. Select the "Add" button.
7. Assign a meaningful name.
8. Specify the required tag data type.
   The "Value" parameter, for example, requires a numeric data type.

9.1.6 Adapt the function list to changed scripts

You use functions that you have defined for global modules in the function list. These functions and associated parameters are referenced.

If you edit used functions after creating the function list, you must manually transfer some changes to the function list:

- Adding of a parameter
- Deleting of a parameter
- Deleting of a function

Note

If you rename functions or parameters, these changes are automatically transferred to the function list.

Requirement

- A function is defined in a global module.
- The function is used in a function list.

Procedure

1. Make at least one of the following changes to the function:
   - Add a parameter
   - Delete a parameter
   - Delete a function
2. Go to the relevant function in the function list.
3. Double-click to apply the change marked in red.

See also

"Scripts" editor (Page 981)

9.2 System functions

9.2.1 LogOff

Description

Logs off the current user on the HMI device.

Use in the function list

Logoff()

The system function "Logoff" has no parameters.

Use in scripts

You can find more information on using the "LogOff" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.LogOff() (Page 6842)

9.2.2 UpdateTag

Description

Reads the current value of the tags with the specified update ID from the PLC. An update ID can be used for several tags.

Note

Tags of the STRUCT data type are not supported.

Use in the function list

UpdateTag (Update ID)

The system function "UpdateTag" has the following parameters:

Parameter	Description
Update ID	Specifies the Update ID.

Use in scripts

You can find more information on using the "UpdateTag" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.UpdateTag() (Page 1390)

9.2.3 InsertElectronicRecord

Description

This system function is used to log user actions in the Audit Trail that are not automatically logged in the Audit Trail. You can also use this system function to require the user to enter an acknowledgment or an electronic signature and a comment for the operator action. A requirement for the use of the system function is that the GMP-compliant configuration is activated under "Runtime settings > GMP".

If you use the "InsertElectronicRecord" system function in a function, the debugger may open if you cancel your input by clicking "Cancel". To compensate for this reaction, you can use the "On Error Resume Next" statement in a function. With this instruction, the next instruction is executed after a runtime error. If you use the "On Error Resume Next" statement, output of system alarms is also suppressed.

Use in the function list

InsertElectronicRecord (name, category, operation type, old value, new value, confirmation type, reason)

The "InsertElectronicRecord" system function has the following parameters:

Parameters	Description
Name	Name of the modified object
Category	Category or class name of the modified object
Operation type	Specifies the type of change. 1: New value 2: Modified value 3: Deleted value
Old value	Value before the change
New value	Value after the change
Confirmation type	Specifies how the action must be confirmed 0 = (None): No confirmation required, an entry is created in the Audit Trail 1 = (Acknowledgement): Acknowledgment, the user must acknowledge the action; an entry is created in the Audit Trail 2 = (Digital Signature): Electronic signature; a dialog window opens in which the user must enter the electronic signature - an entry is created in the Audit Trail
Reason (optional)	May include a comment explaining the reason for the change.

Use in scripts

You can find more information on using the "InsertElectronicRecord" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.InsertElectronicRecord() (Page 1133)

9.2.4 ExecuteReport

Description

Executes a report automatically and independently of the general report cycle. The execution can be triggered through a specific event, for example, change of a tag value, occurrence of a specific message or exceeding a tag limiting value.

Use in the function list

ExecuteReport (name report order)

The system function "ExecuteReport" has the following parameters:

Parameter	Description
Report task name	Specifies the name of the report to be executed.

Use in scripts

You can find more information on using the "ExecuteReport" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ExecuteReport() (Page 1302)

9.2.5 EjectStorageMedium

Description

Ejects an inserted storage medium. The function checks whether the storage medium is currently being accessed. If no current read or write process is taking place, the storage medium can be removed without data loss.

Note

The "EjectStorageMedium" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

The system function only works correctly in runtime if you use the storage medium in question for logging.

Use in the function list

EjectStorageMedium (storage device)

The system function "EjectStorageMedium" has the following parameters:

Parameter	Description
Storage device	System variable name of the storage medium with path specification (e.g. /USB storage card)

Use in scripts

You can find more information on using the "EjectStorageMedium" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.EjectStorageMedium() (Page 1164)

9.2.6 IncreaseTag

Description

Adds the specified value to the tag value: X = X + a

If you configure the system function for events of an alarm without the tag being used in the current screen, it is not ensured that the actual tag value is being used in the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used at the "Process value" property of an object.

Note

The system function is executed with the last known process value. Since this process value cannot be kept up to date in all cases, it is prohibited to write it from multiple sources (e.g. from the HMI device via scripts and from the PLC). This ensures that the value that is changed actually corresponds to the process value.

Use in the function list

IncreaseTag (Tag, Value)

The system function "IncreaseTag" has the following parameters:

Parameter	Description
Tag	Tag to which the specified value is added. The following data types are not supported: Byte, Word, DWord, LWord.
Value	Value to be added.

Note

Converting a value

The system function uses the same tag as input and output values. If you are using this system function to convert a value, follow these steps:

1. Create an auxiliary tag.
2. Assign the tag value to the auxiliary tag with the "SetTagValue" system function.

Use in scripts

You can find more information on using the "IncreaseTag" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.IncreaseTag() (Page 1384)

9.2.7 CreateParameterSet

Description

Creates a new parameter set. with the default values defined in the Engineering System.

Use in the function list

CreateParameterSet (parameter set type, Language, ParameterSetTypeName, Processing status)

The system function "CreateParameterSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set ID (optional)	ID of the parameter set that is created. If an ID is not entered, a unique ID is assigned automatically.
Parameter set name (optional)	Name of the parameter set that is created. If a name is not entered, a unique name is assigned automatically.
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 - Function was canceled.

Using scripts

You can find more information on the "CreateParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.CreateParameterSet() (Page 1245)

9.2.8 CreateScreenshot

Description

Creates and saves a screenshot. The .jpg and .jpeg image formats are supported. If images already exist in the specified file path, they will be overwritten. If the specified file path cannot be accessed, an error message is displayed.

Note

The system function "CreateScreenshot" is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

CreateScreenshot (path of storage medium)

The system function "CreateScreenshot" has the following parameters:

Parameter	Description
Storage media path	Path name of the screenshot.

Use in scripts

You can find more information on using the "CreateScreenshot" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.CreateScreenshot() (Page 1164)

9.2.9 CreateOperatorInputInformation

Description

Creates an alarm of the class "OperatorInputInformation".

Use in the function list

CreateOperatorInputInformation (alarm text, range, parameter value 1, parameter value 2, parameter value 3, parameter value 4, parameter value 5, parameter value 6, parameter value 7)

The system function "CreateSystemInformation" has the following parameters:

Parameter	Description
Alarm text	Specifies the alarm text. The alarm text can only be defined in one language in the system function.
Area (optional)	Specifies the scope of the alarm.
Parameter value 1 (optional)	Value of the first alarm parameter.
Parameter value 2 (optional)	Value of the second alarm parameter.
Parameter value 3 (optional)	Value of the third alarm parameter.
Parameter value 4 (optional)	Value of the fourth alarm parameter.
Parameter value 5 (optional)	Value of the fifth alarm parameter.
Parameter value 6 (optional)	Value of the sixth alarm parameter.
Parameter value 7 (optional)	Value of the seventh alarm parameter.

Use in scripts

You can find more information on using the "CreateOperatorInputInformation" system function in JavaScript functions in the WinCC Unified object model.

See also

Creating alarms with multilingual alarm texts (Page 1015)

SysFct.CreateOperatorInputInformation() (Page 1101)

9.2.10 CreateSystemInformation

Description

Creates an alarm of the class "SystemInformation".

The alarm class "SystemInformation" only has the status "Incoming" and is therefore only displayed under "Logged alarms" in the alarm control and not under "Pending alarms".

Use in the function list

CreateSystemInformation (alarm text, range, parameter value 1, parameter value 2, parameter value 3, parameter value 4, parameter value 5, parameter value 6, parameter value 7)

The system function "CreateSystemInformation" has the following parameters:

Parameter	Description
Alarm text	Specifies the alarm text. The alarm text can only be defined in one language in the system function.
Area (optional)	Specifies the scope of the alarm.
Parameter value 1 (optional)	Value of the first alarm parameter.
Parameter value 2 (optional)	Value of the second alarm parameter.
Parameter value 3 (optional)	Value of the third alarm parameter.
Parameter value 4 (optional)	Value of the fourth alarm parameter.
Parameter value 5 (optional)	Value of the fifth alarm parameter.
Parameter value 6 (optional)	Value of the sixth alarm parameter.
Parameter value 7 (optional)	Value of the seventh alarm parameter.

Use in scripts

You can find more information on using the "CreateSystemInformation" system function in JavaScript functions in the WinCC Unified object model.

See also

Creating alarms with multilingual alarm texts (Page 1015)

SysFct.CreateSystemInformation() (Page 1104)

9.2.11 CreateSystemAlarm

Description

Creates an alarm of the class "SystemAlarm".

Use in the function list

CreateSystemAlarm (alarm text, range, parameter value 1, parameter value 2, parameter value 3, parameter value 4, parameter value 5, parameter value 6, parameter value 7)

The system function "CreateSystemInformation" has the following parameters:

Parameter	Description
Alarm text	Specifies the alarm text. The alarm text can only be defined in one language in the system function.
Area (optional)	Specifies the scope of the alarm.
Parameter value 1 (optional)	Value of the first alarm parameter.
Parameter value 2 (optional)	Value of the second alarm parameter.
Parameter value 3 (optional)	Value of the third alarm parameter.
Parameter value 4 (optional)	Value of the fourth alarm parameter.
Parameter value 5 (optional)	Value of the fifth alarm parameter.
Parameter value 6 (optional)	Value of the sixth alarm parameter.
Parameter value 7 (optional)	Value of the seventh alarm parameter.

Use in scripts

You can find more information on using the "CreateSystemAlarm" system function in JavaScript functions in the WinCC Unified object model.

See also

Creating alarms with multilingual alarm texts (Page 1015)

SysFct.CreateSystemAlarm() (Page 1103)

9.2.12 ExportParameterSets

Description

Exports parameter sets from the parameter set memory to a file.

Use in the function list

ExportParameterSets (parameter set type ID, parameter set ID, file name, overwrite, output status, processing status (optional), generate checksum)

The system function "ExportParameterSets" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set. The following cases are differentiated: • If the parameter set ID is set to 0, all parameter sets available in the memory are exported. • If the specified name or the ID does not exist in the imported file, the execution is canceled. • If the specified name or the ID does not exist in the imported file, this specific parameter set is imported. In the following cases the import is aborted and an alarm appears: • No parameter set available • Name or ID does not exist in the import file
File name	Specifies the file path of the file to be imported. In the following cases the execution is canceled and an alarm is generated: • Invalid file path • Error during file access
Overwrite	Specifies whether the existing file is overwritten: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. An alarm is generated and displayed if an error occurs during file access. This can occur, for example, when the existing file cannot be overwritten even though overwriting is allowed.
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function just executed 4 = Function successfully executed 12 = Function was canceled
Generate checksum	Specifies whether a checksum is generated for the parameter set to be exported: True = Checksum is generated. False = Checksum is not generated.

Use in scripts

You can find more information on using the "ExportParameterSets" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ExportParameterSets() (Page 1248)

9.2.13 GoToPLC

Description

Switches from a diagnostics indicator to the system diagnostics control.

Use in the function list

GoToPLC (Screen object path)

The system function "GoToPLC" has the following parameters:

Parameter	Description
Screen object path	Path to the screen object System Diagnostics Control, for example, "./Screen window_1/Diagnostic control_1"

Using scripts

You can find more information on the "GoToPlc" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GoToPlc() (Page 6835)

9.2.14 ImportParameterSets

Description

Imports parameter sets from a file into the parameter set memory.

Use in the function list

ImportParameterSets (file name, parameter set ID, overwrite, output status, processing status (optional), check checksum)

The system function "ImportParameterSets" has the following parameters:

Parameter	Description
File name	Specifies the file path of the file to be imported. In the following cases the execution is canceled and an alarm is generated: • Invalid file path • Error during file access
Parameter set ID	Specifies the name or the ID of the parameter set. The following cases are differentiated: • If the parameter set ID is set to 0, all parameter sets are imported from the file. • If the name or the ID does not exist in the imported file, the execution is canceled. • If the specified name or the ID does not exist in the imported file, only this specific parameter set is imported. In the following cases the import is aborted and an alarm appears: • Invalid file head. • No parameter set available. • Parameter set name or parameter set ID does not exist in the file.
Overwrite	Specifies whether the values in the memory are overwritten with the values from the import file: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. If the name / ID of the specified parameter set exists, the values in the memory are overwritten with the parameter set values from the import file if overwriting is allowed. If it may not be overwritten, the data in the memory is not renewed.
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was canceled.
Check checksum	Specifies whether the checksum of the import file is verified: True = Checksum is generated. False = Checksum is not generated.

Use in scripts

You can find more information on using the "ImportParameterSets" system function in JavaScript functions in the WinCC Unified object model.

See also

Exporting and importing parameter sets (Page 891)

SysFct.ImportParameterSets() (Page 1254)

9.2.15 InvertBitInTag

Description

Inverts the bit with the specified number in the tag:

- If the bit in the tag has the value 1 (True), it will be set to 0 (False).
- If the bit in the tag has the value 0 (False), it will be set to 1 (True).

After changing the given bit, the system function transfers the entire tag back to the PLC. It is not checked whether other bits in the tags have changed in the meantime.

While the function is running, the operator and controller have only read access to the specified tag until it is transferred to the controller again.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used at the "Process value" property of an object.

Note

The system function is executed with the last known process value. Since this process value cannot be kept up to date in all cases, it is prohibited to write it from multiple sources (e.g. from the HMI device via scripts and from the PLC). This ensures that the value that is changed actually corresponds to the process value.

Use in the function list

InvertBitInTag (Tag, Value)

The system function "InvertBitInTag" has the following parameters:

Parameter	Description
Tag	The tag in which the specified bit is inverted.
Bit number	The number of the bit that is inverted. When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Use in scripts

You can find more information on using the "InvertBitInTag" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.InvertBitInTag() (Page 1384)

9.2.16 IsAlarmJumpPossible

Description

The function checks whether the alarm selected in the alarm control is a process diagnostics alarm. If so, the specified screen object is enabled; otherwise, it is disabled.

Use in the function list

IsAlarmJumpPossible (Alarm control, Screen object path)

The "IsAlarmJumpPossible" system function has the following parameters:

Parameter	Description
Alarm control	Path of the alarm control with selected alarm.
Screen object path	Path of the screen object

Use in scripts

You can find more information on using the "IsJumpableAlarm" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.IsJumpableAlarm() (Page 1561)

9.2.17 LoadParameterSet

Description

Loads parameter sets from the parameter set memory into the edit tag.

Use in the function list

LoadParameterSet (Parameter set type, Parameter set, Output status, Processing status (optional))

The system function "LoadParameterSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 - Function was canceled.

Using scripts

You can find more information on the "LoadParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.LoadParameterSet() (Page 1257)

9.2.18 LoadAndWriteParameterSet

Description

Loads parameter sets from the parameter set memory and writes them to the PLC.

Use in the function list

LoadAndWriteParameterSet (parameter set type ID, parameter set ID, output status, processing status (optional))

The system function "LoadAndWriteParameterSet" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.

Parameter	Description
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 - Function was canceled.

Use in scripts

You can find more information on using the "LoadAndWriteParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

Transferring parameter sets via scripts (Page 879)

Transferring parameter sets (Page 897)

SysFct.LoadAndWriteParameterSet() (Page 1256)

9.2.19 GetDHCPState

Description

Reads out the DHCP setting of the network adapter.

Note

The "GetDHCPState" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

GetDHCPState (Adapter name, Status, IPV6 (optional))

The system function "GetDHCPState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
Status	Tag to which the status is written: 0 = DHCP is disabled. 1 = DHCP is enabled.
IP V6 (optional)	Tag to which the IPv6 address is written.

Use in scripts

You can find more information on using the "GetDHCPState" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetDHCPState() (Page 1166)

9.2.20 GetBrightness

Description

Reads out the brightness value.

Note

The "GetBrightness" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

GetBrightness (Value)

The system function "GetBrightness" has the following parameters:

Parameter	Description
Value	The tag to which the brightness value is written.

Use in scripts

You can find more information on using the "GetBrightness" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetBrightness() (Page 1165)

9.2.21 GetIPV4Address

Description

Reads out the IPv4 settings of the network adapter.

Note

The "GetIPV4Address" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

GetIPV4Address (Adapter name, IP address, Subnet mask, Default gateway (optional), DNS server 1 (optional), DNS server 2 (optional))

The system function "GetIPV4Address" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
IP address	Tag to which the IP address is written.
Subnet mask	Tag to which the subnet mask of the IPv4 address is written.
Default gateway (optional)	Tag to which the IP address of the default gateway is written.
DNS server 1 (optional)	Tag to which the IP address of DNS server 1 is written.
DNS server 2 (optional)	Tag to which the IP address of DNS server 2 is written.

Use in scripts

You can find more information on using the "GetIPV4Address" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetIPV4Address() (Page 1168)

9.2.22 GetNetworkInterfaceState

Description

Reads out the status of the network adapter.

Note

The "GetNetworkInterfaceState" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

GetNetworkInterfaceStatus (Adapter name, Status)

The system function "GetNetworkInterfaceState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 addresses PROFINET (X1) Port 0 • X 1.0 addresses PROFINET (X1) Port 0 • X 1.1 addresses PROFINET (X1) Port 1 • X2 = addresses Ethernet (X2) • Manual input
Status	Tag to which the state of the network adapter is written: 0 = Network adapter is disabled. 1 = Network adapter is enabled.

Use in scripts

You can find more information on using the "GetNetworkInterfaceState" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetNetworkInterfaceState() (Page 1170)

9.2.23 ReadParameterSet

Description

Transfers the values of the recipe data set loaded in the controller to the corresponding edit tags.

Use in the function list

ReadParameterSet (Parameter set type, Output status, Processing status (optional))

The system function "ReadParameterSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was canceled.

Using scripts

You can find more information on the "ReadParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ReadParameterSet() (Page 1261)

9.2.24 ReadParameterSetName

Description

Reads the name of the specified parameter set.

Use in the function list

ReadParameterSetName (Parameter set type ID, Parameter set ID, Language, ParameterSetName, Processing status)

The system function "ReadParameterSetName" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set.
Language	Specifies the language of the ID that is to be read.
Parameter set name	The tag to which the name of the parameter set is written.
Processing status	Indicates the execution status of a function: 2 = Function is currently being executed. 4 = Function successfully executed. 12 - Function was canceled

Use in scripts

You can find more information on using the "GetParameterSetName" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetParameterSetName() (Page 1250)

9.2.25 ReadParameterSetTypeName

Description

Reads the name of the specified parameter set type.

Use in the function list

ReadParameterSetTypeName (Parameter set type ID, Language, ParameterSetTypeName, Processing status)

The system function "ReadParameterSetTypeName" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Language	Specifies the language of the ID that is to be read.
Parameter set type name	The tag to which the name of the parameter set type is written.
Processing status	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was canceled.

Using scripts

You can find more information on the "GetParameterSetTypeName" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetParameterSetTypeName() (Page 1252)

9.2.26 GetSmartServerState

Description

Returns the activation state of the Smart server.

Note

The "GetSmartServerState" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

GetSmartServerState (Status)

The system function "GetSmartServerState" has the following parameters:

Parameter	Description
Status	Tag to which the activation status of the Smart Server is written: True = Smart Server is enabled. False = Smart Server is disabled.

Use in scripts

You can find more information on using the "GetSmartServerState" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.GetSmartServerState() (Page 1171)

9.2.27 ReadAndSaveParameterSet

Description

Reads a parameter set from the PLC and writes the parameter set to the parameter set memory.

Use in the function list

ReadAndWriteParameterSet (parameter set type ID, parameter set ID, overwrite, output status, processing status (optional))

The system function "ReadAndWriteParameterSet" has the following parameters:

Parameter	Description
Parameter set type ID	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set ID	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, a new parameter set is created. If the name or the ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if hmiOverwrite.Enabled is set. If hmiOverwrite.Disabled is set, the data in the memory is not replaced.
Overwrite	Specifies whether the values in the memory are overwritten with the values from the import file: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. The following cases are differentiated: • If the name / ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if overwriting is allowed. • If overwriting is not allowed, the data in the memory is not replaced. The process tag is updated to the status of the system function, if configured accordingly.
Output status	Specifies the output status: True = Alarms are output. False = Alarms are not output. If the output status is set to "True", alarms are generated and displayed (if configured).
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function was successfully executed. 12 = Function cancelled.

Use in scripts

You can find more information on using the "ReadAndSaveParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

Transferring parameter sets via scripts (Page 879)

Transferring parameter sets (Page 897)

SysFct.ReadAndSaveParameterSet() (Page 1259)

9.2.28 ClearAlarmLog

Description

Deletes all recordings from the specified alarm log.

Note

No backup

Note that no automatic backup is performed before the execution of the function.

Use in the function list

ClearAlarmLog (Log name)

The system function "ClearAlarmLog" has the following parameters:

Parameter	Description
Log name	Name of the alarm log from which the entries are deleted.

Use in scripts

You can find more information on using the "ClearAlarmLog" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ClearAlarmLog() (Page 1131)

9.2.29 DeleteParameterSet

Description

Deletes a parameter set.

Use in the function list

DeleteParameterSet (Parameter set type, Parameter set, Output status, Processing status (optional))

The system function "DeleteParameterSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the type of parameter set from which the parameter set will be deleted.
Parameter set	Specifies the name or the ID of the parameter set that is being deleted.
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function was successfully executed. 12 - Function was canceled.

Use in scripts

You can find more information on using the "DeleteParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.DeleteParameterSet() (Page 1247)

9.2.30 ClearTagLog

Description

Deletes all data records in the specified data log.

Note

No backup!

Note that no automatic backup is performed before execution of the function!

Use in the function list

ClearTagLog (Log name)

The system function "ClearTagLog" has the following parameters:

Parameter	Description
Log name	Name of the data log from which all entries are deleted.

Use in scripts

You can find more information on using the "ClearTagLog" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ClearTagLog() (Page 1331)

9.2.31 OpenScreenInPopup

Description

Opens a screen in a popup window.

Use in the function list

OpenScreenInPopup (Popup window name, Screen name, Close when open, Header, Left, Top, Hide close button, Parent screen path)

The system function "OpenScreenInPopup" has the following parameters:

Parameter	Description
Popup window name	Name of the popup window. The name must be unique within the parent screen and is required to close the pop-up window.
Screen name	Name of the screen that is to be opened in the pop-up window.
Close when opened	True = If the pop-up window is open when the function is called, it is closed. False = If the popup window is open when the function is called, it remains open.
Header	Specifies the window title of the popup window.
Left	Defines the window position as offset from the left margin.
Top	Defines the window position as offset from the top margin.
Hide close button	True = The "Close" button is not displayed. False = The "Close" button is displayed.
Parent screen path (optional)	Path of the parent screen. With this parameter, you specify whether the pop-up window is closed in case of a screen change or in case of a screen change in a screen window. If this value is not defined, the pop-up window is global. The pop-up window remains open until it is closed manually, by means of a function call or by exiting runtime.

Use in scripts

You can find more information on using the "OpenScreenInPopup" system function in JavaScript functions in the WinCC Unified object model.

See also

Opening and closing a screen in a pop-up window (Page 1017)

SysFct.OpenScreenInPopup() (Page 6844)

9.2.32 OpenScreenWithNumberInPopup

Description

Opens a screen in a popup window.

Use in the function list

OpenScreenWithNumberInPopup (Popup window name, Screen number, Close when open, Header, Left, Top, Hide close button, Parent screen path)

The system function "OpenScreenWithNumberInPopup" has the following parameters:

Parameter	Description
Popup window name	Name of the popup window. The name must be unique within the parent screen and is required to close the pop-up window.
Screen number	Unique number (> 0) of the screen that will be loaded into the popup window.
Close when opened	True = If the pop-up window is open when the function is called, it is closed. False = If the popup window is open when the function is called, it remains open.
Header	Specifies the window title of the popup window.
Left	Defines the window position as offset from the left margin.
Top	Defines the window position as offset from the top margin.
Hide close button	True = The "Close" button is not displayed. False = The "Close" button is displayed.
Parent screen path (optional)	Path of the parent screen. With this parameter, you specify whether the pop-up window is closed in case of a screen change or in case of a screen change in a screen window. If this value is not defined, the pop-up window is global. The pop-up window remains open until it is closed manually, by means of a function call or by exiting runtime.

Use in scripts

You can find more information on using the "OpenScreenByNumberInPopup" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.OpenScreenByNumberInPopup() (Page 6842)

9.2.33 OpenViewGRAPHByBlock

Description

Switches from any screen object to the PLC code view.

Use in the function list

OpenViewGRAPHByBlock (PLC name, GRAPH instance name, step number, screen object path)

The "OpenGRAPHViewerGraphByBlock" system function has the following parameters:

Parameter	Description
PLC name	Indicates the name of the PLC.
GRAPH instance name	Instance name of the GRAPH block to be displayed.
Step number	Number of the step to be displayed.
Screen object path	Path of the screen object PLC code view. For example, "./Screen window_1/ PlcCodeViewer control_1"

Using scripts

You can find more information on the "OpenViewerGraphByBlock" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.OpenViewerGraphByBlock() (Page 1566)

9.2.34 OpenGRAPHViewFromOverview

Description

Switches from the GRAPH overview to the PLC code view.

Use in the function list

OpenGRAPHViewFromOverview (Object path to GRAPH overview, Object path to PLC code view)

The system function "OpenGRAPHViewFromOverview" has the following parameters:

Parameter	Description
Object path to GRAPH overview	Path to the screen object GRAPH overview, from which switching is to take place.
Object path to PLC code display	Path to the screen object PLC code view.

Using scripts

You can find more information on the "OpenViewerGraphFromOverview" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.OpenViewerGraphFromOverview() (Page 1567)

9.2.35 OpenPLCCodeViewByAlarm

Description

Opens the corresponding block in the PLC code view according to the selected alarm in the alarm control.

Use in the function list

OpenPLCCodeViewByAlarm (Alarm control, PLC code view)

The "OpenPLCCodeViewByAlarm" system function has the following parameters:

Parameter	Description
Alarm control	Path of the alarm control
PLC code view	Path to the PLC code view

Use in scripts

You can find more information on the "OpenPlcCodeViewFromAlarm" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.OpenPlcCodeViewFromAlarm() (Page 1563)

9.2.36 ResetBitInTag

Description

Sets the bit with the specified number to 0 in the tag (False).

After changing the given bit, the system function transfers the entire tag back to the PLC. It is not checked whether other bits in the tags have changed in the meantime. Operator and PLC have read-only access to the indicated tag until it is transferred back to the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used at the "Process value" property of an object.

Note

The system function is executed with the last known process value. Since this process value cannot be kept up to date in all cases, it is prohibited to write it from multiple sources (e.g. from the HMI device via scripts and from the PLC). This ensures that the value that is changed actually corresponds to the process value.

Use in the function list

ResetBitInTag (Tag, Value)

The system function "ResetBitInTag" has the following parameters:

Parameter	Description
Tag	The tag in which a bit is set to 0 (False).
Bit number	The number of the bit that is set to 0 (False). When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Use in scripts

You can find more information on using the "ResetBitInTag" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ResetBitInTag() (Page 1385)

9.2.37 ShiftAndMask

Description

The system function converts the input bit pattern of the source into an output bit pattern of the target. This involves bit shifting and masking. An integer number serves as bit mask, with whose bit pattern the shifted input bit pattern is multiplied. The shifted input bit pattern is multiplied by the bit mask, with bit-by-bit logical AND operation. The result has a decimal value and is stored in the target.

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

You can enter the bit mask in three different ways:

- Hexadecimal: First, enter "0h" or "0H" as prefix, followed by an optional space for better readability. Then group the bit pattern in blocks of four, for example (0000)(1001)(1010)(1110), and set each block in hexadecimal code: (0)(9)(A)(E). Only the characters 0-9, A-F, a-f are permitted for the input: "0h 09AE".
- Binary: First, enter "0b" or "0B" as prefix, followed by an optional space for better readability. Then group the binary bit pattern into blocks of four, for example 0000 1001 1010 1110 with spaces in between as a check. Only the characters "0" or "1" are permitted for the input: "0b 0000 1001 1010 1110".
- Decimal: Enter the value directly without prefix, for example "2478".

Use in the function list

ShiftAndMask (Source, Target, Bits to shift, Bit pattern)

The system function "ShiftAndMask" has the following parameters:

Parameter	Description
Source	The Source parameter includes the input bit pattern. Integer-type tags, e.g. "Byte", "Char", "Int", "UInt", "Long" and "ULong", are permitted. Example: The source of the 16-bit integer type has the current value 72: 0000000001001000.
Target	The output-bit pattern is stored in the Target. Integer type tags, e.g. "Byte", "Char", "Int", "UInt", "Long" and "ULong" are permitted.
Bits to shift	Number of bits by which the input bit pattern is shifted right. A negative value shifts the input bit pattern to the left. Example: "Bits to shift" has the value "+3". The input bit pattern is shifted right by three bits when the system function is called: 0000000000001001. Bits to the left are padded with "0". Three bits are truncated on the right. The new decimal value is "9".
Bit pattern	An integer serves as bit mask. The bit pattern is used to multiply the shifted input bit pattern.

Note

Note the following:

- Source and target have the same number of bits. When the source and target have a different number of bits, using the system function in the target can result in a violation of the value range.
- The number of the bits to shift is smaller than the number of bits of source and target.
- Bit pattern has no more bits than source and target.
- The left bit is "1" if the source has a signed Integer data type with the sign "-". This sign bit is padded with "0" when the bits are shifted right. The sign changes to "+".

Use in scripts

You can find more information on using the "ShiftAndMask" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ShiftAndMask() (Page 1388)

9.2.38 ClosePopup

Description

Closes a pop-up window dynamically during runtime.

Use in the function list

ClosePopup (popup window path)

The system function "ClosePopup" has the following parameters:

Parameter	Description
Popup window path	Specifies the path to the popup window to be closed.

Use in scripts

You can find more information on using the "ClosePopup" system function in JavaScript functions in the WinCC Unified object model.

See also

Opening and closing a screen in a pop-up window (Page 1017)

SysFct.ClosePopup() (Page 6841)

9.2.39 WriteParameterSet

Description

Writes the values of the edit tag to the PLC.

Use in the function list

WriteParameterSet (Parameter set type, Output status, Processing status (optional))

The system function "WriteParametersSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was canceled.

Using scripts

You can find more information on the "WriteParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.WriteParameterSet() (Page 1266)

9.2.40 WriteManualValue

Description

Assigns a new value to the specified logging tag. The associated time stamp is transferred in this process.

Use in the function list

WriteManualValue (Logging tag name, Value, Time stamp)

The system function "WriteManualValue" has the following parameters:

Parameter	Description
Logging tag name	Logging tag to which the specified value is assigned.
Value	The value assigned to the specified logging tag.
Time stamp	The time stamp assigned to the specified value.

Use in scripts

You can find more information on using the "WriteManualValue" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.WriteManualValue() (Page 1332)

9.2.41 SetBitInTag

Description

Sets the bit with the specified number to 1 in the tag (True).

After the specified bit is changed, the system function transfers the entire tag back to the PLC. It is not checked whether other bits in the tags have changed in the meantime. HMI device and PLC only have read access to the specified tag until it is transferred back to the PLC.

For the system function to be executed, the value of the tags must be current and valid, and the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- The connection to the PLC is set up.
- The acquisition mode of the tags is "Cyclic in operation".
- The tag is used at the "Process value" property of an object.

Note

The system function is executed with the last known process value. Since this process value cannot be kept up to date in all cases, it is prohibited to write it from multiple sources (e.g. from the HMI device via scripts and from the PLC). This ensures that the value that is changed actually corresponds to the process value.

Data types not supported

The following data types are not supported by the function:

- UINT

Use in the function list

SetBitInTag (Tag, Value)

The "SetBitInTag" system function has the following parameters:

Parameter	Description
Tag	The tag in which a bit is set to 1 (True).
Bit number	The number of the bit that is set to 1 (True). When this system function is used in a user function, the bits in the given tag will be counted from right to left independent of the controller that is used. Numbering starts with 0.

Use in scripts

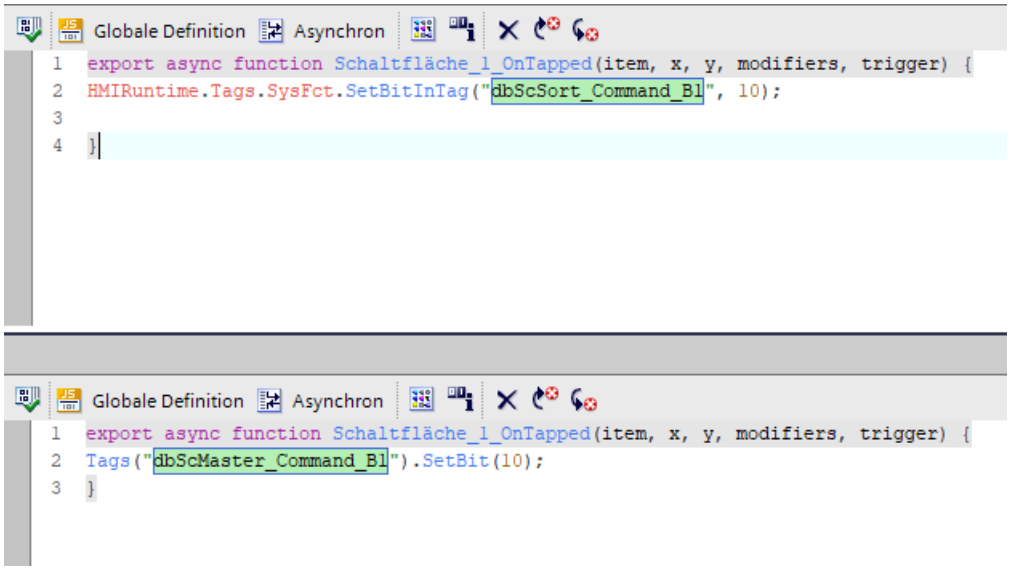
You can find more information on using the "SetBitInTag" system function in JavaScript functions in the WinCC Unified object model.

Use of "SetBitInTag" in structure tags

If "SetBitInTag" is used at a structure tag, the result is that after setting and resetting individual bits, all previously changed bits are overwritten when they are set again. This occurs, for example, as a result of an event of a button.

To prevent this, follow these steps:

1. Convert the function at the event to JavaScript.



2. Replace the "SetBitInTag" system function with the "SetBit()" method of the "Tags" object.

See also

Tag.SetBit() (Page 1351)

SysFct.SetBitInTag() (Page 1386)

9.2.42 SetDHCPState

Description

Changes the DHCP setting of the network adapter.

Note

The "SetDHCPState" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

SetDHCPState (Adapter name, Enabled, IP V6 (optional))

The system function "SetDHCPState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
Enabled	Defines the DHCP setting of the network adapter: 0 = DHCP is disabled. 1 = DHCP is enabled.
IP V6 (optional)	Tag to which the IPv6 address is written.

Use in scripts

You can find more information on using the "SetDHCPState" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetDHCPState() (Page 1173)

9.2.43 SetPropertyValue

Description

Assigns a new value to the specified property of the screen item.

Note

Depending on the type of the object property, you can use this system function to assign strings and numbers.

Use in the function list

SetPropertyValue (Screen item path, Screen item property name, Value)

The system function "SetPropertyValue" has the following parameters:

Parameter	Description
Screen object path	Path of the screen item whose property is changed.
Screen object property name	Name of the property that will be changed. The property name can be copied with selected screen item to the Inspector window under "Properties > Properties": 1. Right-click the name of the property. 2. Select "Copy property name".
Value	The value assigned to the property.

Use in scripts

You can find more information on using the "SetPropertyValue" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetPropertyValue() (Page 6849)

9.2.44 SetBrightness

Description

Assigns a new value to the brightness of the display.

The value for the system function "SetBrightness" can be set between 0% and 100%. The set value is transferred to the HMI device. The brightness settings on the HMI device can be viewed and edited in "Start Center > Settings > Display". The HMI devices support a brightness setting between 10% and 100%.

If the system function "SetBrightness" is assigned a value of 0%, the display of the HMI device is switched off by default in runtime. If the operator touches the display, the display switches to the previous brightness setting.

If the system function "SetBrightness" is assigned a value between 1% and 10% and the operator opens the display settings in the Start Center, brightness is reset to 10%.

Note

The "SetBrightness" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Note

The configuration that is set in the Control Panel / Start Center will be reestablished when you restart the HMI device.

Use in the function list

SetBrightness (value)

The system function "SetBrightness" has the following parameters:

Parameter	Description
Value	The new value for the brightness of the display.

Use in scripts

You can find more information on using the "SetBrightness" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetBrightness() (Page 1172)

9.2.45 SetIPV4Address

Description

Changes the IPv4 settings of the network adapter.

Note

The "SetIPV4Address" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

SetIPV4Address (Name adapter, IP address, Subnet mask, Standard gateway (optional), DNS Server 1 (optional), DNS Server 2 (optional))

The system function "SetIPV4Address" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = Static network adapter name 1 • X2 = Static network adapter name 2 • Manual input
IP address	Specifies the IP address.
Subnet mask	Subnet mask of the IPv4 network.
Default gateway (optional)	IP address of the default gateway.
DNS server 1 (optional)	IP address of DNS server 1.
DNS server 2 (optional)	IP address of DNS server 2.

Use in scripts

You can find more information on using the "SetIPV4Address" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetIPV4Address() (Page 1174)

9.2.46 SetNetworkInterfaceState

Description

Changes the state of the network adapter.

Note

The "SetNetworkInterfaceState" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

SetNetworkInterfaceState (Adapter name, Enabled)

The system function "SetNetworkInterfaceState" has the following parameters:

Parameter	Description
Adapter name	Specifies the name of the network adapter. The following entries are possible: • X1 = addresses PROFINET (X1) Port 0. • X1.0 = addresses PROFINET (X1) Port 0. • X1.1 = addresses PROFINET (X1) Port 1. • X2 = addresses Ethernet (X2). • Manual input
Enabled	Specifies the state of the network adapter: 0 = Network adapter is disabled. 1 = Network adapter is enabled.

Note

The Panel must be restarted after a change in the status of X1.0 or X1.1.

Use in scripts

You can find more information on using the "SetNetworkInterfaceState" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetNetworkInterfaceState() (Page 1176)

9.2.47 SetLanguage

Description

Toggles the language on the HMI device. All configured texts and system events are displayed on the HMI device in the newly set language.

Use in the function list

SetLanguage (LCID)

The system function "SetLanguage" has the following parameters:

Parameter	Description
LCID	LCID of the language set on the HMI device. Specify the language ID, e.g. 1031 for German - Standard, 1033 for English - USA. You can find an overview of all languages under: "https://docs.microsoft.com/de-de/deployoffice/office2016/language-identifiers-and-optionstate-id-values-in-office-2016".

Use in scripts

You can find more information on using the "SetLanguage" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetLanguage() (Page 6845)

9.2.48 SetSmartServerState

Description

Allows you to enable or disable the smart server.

Note

The "SetSmartServerState" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

SetSmartServerState (Status)

The "SetSmartServerState" system function has the following parameters:

Parameter	Description
Status	Status to which the Smart Server is to be set: True = Smart Server is enabled. False = Smart Server is disabled.

Use in scripts

You can find more information on using the "SetSmartServerStart" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetSmartServerState() (Page 1177)

9.2.49 SetTagValue

Description

Assigns the specified tag a new value.

Depending on the tag type, you use this system function to assign strings and numbers.

Note

The "SetTagValue" system function is only executed after a connection has been established.

Use in the function list

SetTagValue (Tag, Value)

The system function "SetTagValue" has the following parameters:

Parameter	Description
Tag	Tag to which the specified value is assigned.
Value	The value that is assigned to the specified variable.

Use in scripts

You can find more information on using the "SetTagValue" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetTagValue() (Page 1387)

9.2.50 SetConnectionMode

Description

The specified connection is established or disconnected.

Use in the function list

SetConnectionMode (Connection name, Activated)

The "SetConnectionMode" system function has the following parameters:

Parameter	Description
Connection name	The PLC whose connection to the HMI device is established or disconnected. You specify the name of the PLC in the connection editor.
Enabled	0 = Offline: Connection is terminated. 1 = Online: Connection is established.

Use in scripts

You can find more information on using the "SetConnectionMode" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SetConnectionMode() (Page 1152)

9.2.51 SaveParameterSet

Description

Saves the current values of the edit tag of a parameter set to the memory of the HMI device.

Use in the function list

SaveParameterSet (Parameter set type, Parameter set, Output status, Overwrite, Processing status (optional))

The system function "SaveParametersSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, a new parameter set is created.
Overwrite	Specifies whether an existing parameter set is overwritten: 0 = Overwriting is not allowed. 1 = Overwriting is allowed. The following cases are differentiated: • If the name / ID of the specified parameter set exists, the values in the PLC are overwritten with the parameter set values in the memory if overwriting is allowed. • If overwriting is not allowed, the data in the memory is not replaced. The process tag is updated to the status of the system function, if configured accordingly.

Parameter	Description
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was canceled.

Using scripts

You can find more information on the "SaveParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.SaveParameterSet() (Page 1264)

9.2.52 StartProgram

Description

Starts the specified program on the HMI device.

The runtime software continues running in the background.

Alarms continue to be output and process values continue to be updated.

When the given application is exited, the screen which was active during the performance of the system function is displayed on the HMI device.

This system function is used, for example, to edit recipe data records in MS Excel on the HMI device.

The function is supported by both the Windows and Linux systems.

Note

On a SIMATIC Unified PC, this system function can only be used to start applications that do not have a user interface.

Use in the function list

StartProgram (Program name, Program parameters, Display mode, Wait for end of program, Result (optional))

The system function "StartProgram" has the following parameters:

Parameter	Description
Program name	Name and path of the program which is started. Upper and lower case are taken into account in this parameter.
Program parameters	The parameters you transfer at the start of the program, for example a file that is opened after the start of the program. You can find additional information on the necessary parameters in the documentation of the program to be started.
Display mode	Determines how the program window is displayed on the HMI device. This function has no effect on Linux systems.
Waiting for program to end	Determines whether there is a change back to the project after the called up program has ended: 0 = No change to project. 1 = Change to project.
Result (optional)	Contains data that can be written to the standard output from an external application.

Note

If the path for the program name contains spaces, the program can only be started correctly if the path is specified in quotation marks, e.g. "C:\Program Files\START\start.exe".

Use in scripts

You can find more information on using the "StartProgram" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.StartProgram() (Page 1179)

Win32 Microsoft (https://docs.microsoft.com/en-us/locale/?target=https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-showwindow)

9.2.53 StopRuntime

Description

Ends the runtime software and the project running on the HMI device.

The system function can be used with or without parameters. If no parameters are specified, the system function remains undefined and stops the running project. In this case, the HMI device is not restarted.

Note

All functions after "StopRuntime" are not executed.

Use in the function list

StopRuntime (StopRuntime, stop runtime and restart operating system)

The "StopRuntime" system function can be used with or without the following parameter:

Parameter	Description
Mode (optional)	Sets the type of termination: 0 = Ends runtime. 1 = Ends runtime and restarts the device.

Use in scripts

You can find more information on using the "StopRuntime" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.StopRuntime() (Page 1181)

9.2.54 LookUpText

Description

Identifies a list entry from a text list. The result depends on the value of the list entry and the selected runtime language. The result is saved to a tag of data type "String".

Use in the function list

CallText (output text, index, LCID, text list name)

The system function "LookUpText" has the following parameters:

Parameter	Description
Output text	The tag to which the result is written.
Index	The tag that defines the value of the list entry.

Parameter	Description
LCID	LCID of the language set on the HMI device. Specify the language ID, e.g. 0x0409 for English - USA, 0x0007 for German - Standard. You can find an overview of all languages under: "https://msdn.microsoft.com/en-us/library/cc233982.aspx".
Text list name	Defines the text list. The list entry is read from the text list.

Use in scripts

You can find more information on using the "LookUpText" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.LookUpText() (Page 1312)

9.2.55 RenameParameterSet

Description

Changes the name of a parameter set.

Use in the function list

RenameParameterSet (Parameter set type, Parameter set, NewParameterSetName, Output status, Processing status (optional))

The system function "RenameParametersSet" has the following parameters:

Parameter	Description
Parameter set type	Specifies the name or the ID of the parameter set type. If the name or ID of the parameter set type does not exist, execution is terminated.
Parameter set	Specifies the name or the ID of the parameter set. If the name or ID of the parameter set does not exist, execution is terminated.
New parameter set name	Specifies the new name of the parameter set in the current Runtime language.
Output status	Defines the output status: True = Alarms are output. False = Alarms are not output.
Processing status (optional)	Indicates the execution status of a function: 2 = Function is being executed. 4 = Function successfully executed. 12 = Function was canceled.

Using scripts

You can find more information on the "RenameParameterSet" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.RenameParameterSet() (Page 1262)

9.2.56 ToggleGRAPHViewerMode

Description

Switches to the GRAPH-Viewer Mode of the PLC code view.

Use in the function list

ToggleGRAPHViewerMode (object path to PLC code view)

The system function "ToggleGRAPHViewerMode" has the following parameters:

Parameter	Description
Object path to PLC code display	Path to the PLC code view.

Using scripts

You can find more information on the "ToggleGRAPHViewerMode" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ToggleGRAPHViewerMode() (Page 1569)

9.2.57 ToggleNetworkDisplay

Description

Switches to the network display of the PLC code view.

Use in the function list

ToggleNetworkDisplay (Object path to PLC code view)

The system function "ToggleNetworkDisplay" has the following parameters:

Parameter	Description
Object path to PLC code display	Path to the PLC code view.

Using scripts

You can find more information on the "ToggleNetworkDisplay" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ToggleNetworkDisplay() (Page 1570)

9.2.58 ToggleLanguage

Description

This function allows you to change the runtime language in order to display language-dependent texts correctly on the user interface. The conversion takes place according to the runtime language configuration.

Use in the function list

ToggleLanguage()

The system function "ToggleLanguage" has no parameters.

Use in scripts

You can find more information on using the "ToggleLanguage" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ToggleLanguage() (Page 6849)

9.2.59 ZoomIn

Description

Zooms in the view of the network in the network area of a PLC Code View.

Use in the function list

Zoom in (Object path to PLC code view)

The system function "ZoomIn" has the following parameters:

Parameter	Description
Object path to PLC code display	Path to the PLC code view.

Using scripts

You can find more information on the "ZoomIn" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ZoomIn() (Page 1570)

9.2.60 ZoomOut

Description

Zooms out of the view of the network in the network area of a PLC code view.

Use in the function list

ZoomOut (Object path to PLC code view)

The system function "ZoomOut" has the following parameters:

Parameter	Description
Object path to PLC code display	Path to the PLC code view.

Using scripts

You can find more information on the "ZoomOut" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ZoomOut() (Page 1571)

9.2.61 DecreaseTag

Description

Subtracts the given value from the tag value: X = X - a

If you configure the system function for events of an alarm without the tag being used in the current screen, it is not ensured that the actual tag value is being used in the PLC.

For the system function to be executed, the value of the tags must be current and valid, which means the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used at the "Process value" property of an object.

Note

The system function is executed with the last known process value. Since this process value cannot be kept up to date in all cases, it is prohibited to write it from multiple sources (e.g. from the HMI device via scripts and from the PLC). This ensures that the value that is changed actually corresponds to the process value.

Use in the function list

DecreaseTag (Tag, Value)

The system function "DecreaseTag" has the following parameters:

Parameter	Description
Tag	Tag from which the specified value is subtracted. The following data types are not supported: Byte, Word, DWord, LWord.
Value	Value to be subtracted.

Note

Converting a value

The system function uses the same tag as input and output values. If you are using this system function to convert a value, follow these steps:

1. Create an auxiliary tag.
2. Assign the tag value to the auxiliary tag with the "SetTagValue" system function.

Use in scripts

You can find more information on using the "DecreaseTag" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.DecreaseTag() (Page 1383)

9.2.62 ChangeScreen

Description

Loads a new screen into a screen window.

Note

The function list is updated when the screen changes. The functions after "ChangeScreen" are therefore not executed.

Always execute "ChangeScreen" as the last function.

Use in the function list

ChangeScreen (Screen name, Screen window path)

The system function "ChangeScreen" has the following parameters:

Parameter	Description
Screen name	Name of the screen to which you change.
Screen window path	Path of the screen window or base screen that is displayed after the change has been completed. You can find additional information at "Addressing screen windows".

Use in scripts

You can find more information on using the "ChangeScreen" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ChangeScreen() (Page 6837)

9.2.63 ChangeScreenAsync

Description

Switches the screen in a screen window. Switching only takes place after the current screen has been fully loaded.

Principle

ChangeScreenAsync (Screen name, Screen window path)

The system function "ChangeScreenAsync" has the following parameters:

Parameter	Description
Screen name	Specifies the name of the new screen.
Screen window path	Specifies the path of the screen window in which the screen is to be switched, for example, "Screen_window_1"

Using scripts

You can find more information on the "ChangeScreenAsync" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ChangeScreenAsync() (Page 6837)

9.2.64 ChangeScreenAsyncWithNumber

Description

Switches the screen asynchronously in a screen window. Switching only takes place after the current screen has been fully loaded.

Principle

ChangeScreenAsyncWithNumber (screen number, screen window path)

The system function "ChangeScreenAsyncWithNumber" has the following parameters:

Parameter	Description
Screen number	Unique number (> 0) of the new screen to which you change.
Screen window path	Path of the top-level screen window or screen in which the screen is to be switched, for example, "Screen_window_1"

Using scripts

You can find more information on the "ChangeScreenAsyncByNumber" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ChangeScreenAsyncByNumber() (Page 6838)

9.2.65 ChangeScreenWithNumber

Description

Performs a screen change to the specified screen window.

Note

The function list is updated when the screen changes. The functions after "ChangeScreenWithNumber" are therefore not executed.

Always execute "ChangeScreenWithNumber" as the last function.

Use in the function list

ChangeScreenWithNumber (screen number, screen window path)

The system function "ChangeScreenWithNumber" has the following parameters:

Parameter	Description
Screen number	Unique number (> 0) of the new screen to which you change.
Screen window path	Path of the top-level screen window or screen in which the screen is to be switched, for example, "Screen_window_1"

Use in scripts

You can find more information on using the "ChangeScreenByNumber" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ChangeScreenByNumber() (Page 6840)

9.2.66 ChangeConnection

Description

Changes the connection parameters of an HMI connection. The following parameters can be changed: The IP address, the slot number and the rack number. Because the function is executed synchronously, the return value returns an error code that provides immediate information about the cause of the error. The error code can only be read if the function is called via a script.

Note

Change of function parameters after a function call

With the execution of the function you change the function parameters. The new connection may not be active yet at this point.

Note

Usage on devices of the S7 Plus PLC family

For devices of the S7 Plus PLC family (PLCs 15xx and 12xx) it is not possible to change the slot or the rack. The system function cannot be executed if parameters for slot or rack are set.

Use in the function list

ChangeConnection (Connection name, IP V4 address, Slot (optional), Rack (optional))

The "ChangeConnection" system function has the following parameters:

Parameter	Description
Connection name	Indicates the name of the connection.
IP V4 address	Specifies the IPv4 address. Example: 192.169.153.45
Slot (optional)	Specifies the slot number. Permitted values from 1 to 32.
Rack (optional)	Specifies the rack number. Permitted values from 0 to 7.

Use in scripts

You can find more information on using the "ChangeConnection" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ChangeConnection() (Page 1150)

9.2.67 Next

Description

Goes to the next network in a PLC code view.

Use in the function list

Next (Object path to PLC code view)

The system function "Next" has the following parameters:

Parameter	Description
Object path to PLC code display	Path to the PLC code view.

Using scripts

You can find more information on the "Next" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.Next() (Page 1561)

9.2.68 ShowControlPanel

Description

- Hides or shows the Control Panel.
- Opens an applet in runtime.

Note

The "ShowControlPanel" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

ShowControlPanel (Home)

The "ShowControlPanel" system function has the following parameter:

Parameter	Description
Home page	Specifies the applet to be displayed: "AppletName"

Use in scripts

You can find more information on using the "ShowControlPanel" system function in JavaScript functions in the WinCC Unified object model.

List of available applets

List of applet names that are available to the system function:

Display page	Applet name
Panel information	SystemProperties.PanelInformation
Display	SystemProperties.Display
Screensaver	SystemProperties.Screensaver
Reboot	SystemProperties.Reboot
Network settings	NetworkandInternet.NetworkSettings

See also

SysFct.ShowControlPanel() (Page 1178)

9.2.69 ShowSoftwareVersion

Description

Hides or shows the version number of the Runtime software.

Use this system function if during servicing, for example, you required the version of the runtime software used.

Note

The "ShowSoftwareVersion" system function is only available for WinCC Unified Comfort Panel. The system outputs a compiler warning if the function is used through manual input or through a device replacement in SIMATIC WinCC Unified PC.

Use in the function list

ShowSoftwareVersion ()

The system function "ShowSoftwareVersion" has no parameters.

Use in scripts

You can find more information on using the "ShowSoftwareVersion" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.ShowSoftwareVersion() (Page 1178)

9.2.70 Previous

Description

Navigates to the previous network in a PLC code view.

Use in the function list

Previous (Object path to PLC code view)

The "Back" system function has the following parameter:

Parameter	Description
Object path to PLC code display	Path to the PLC code view.

Using scripts

You can find more information on the "Previous" system function in JavaScript functions in the WinCC Unified object model.

See also

SysFct.Previous() (Page 1567)

Programming scripts 10

10.1 Runtime scripting

10.1.1 Introduction to runtime scripting

Area of application

You use Runtime Scripting in WinCC for the following tasks in runtime:

- Dynamization of properties
- Triggering functions for an event

Runtime Scripting uses JavaScript as the programming language. Runtime Scripting is supported at the following objects:

- Screen
- Screen object
- Task

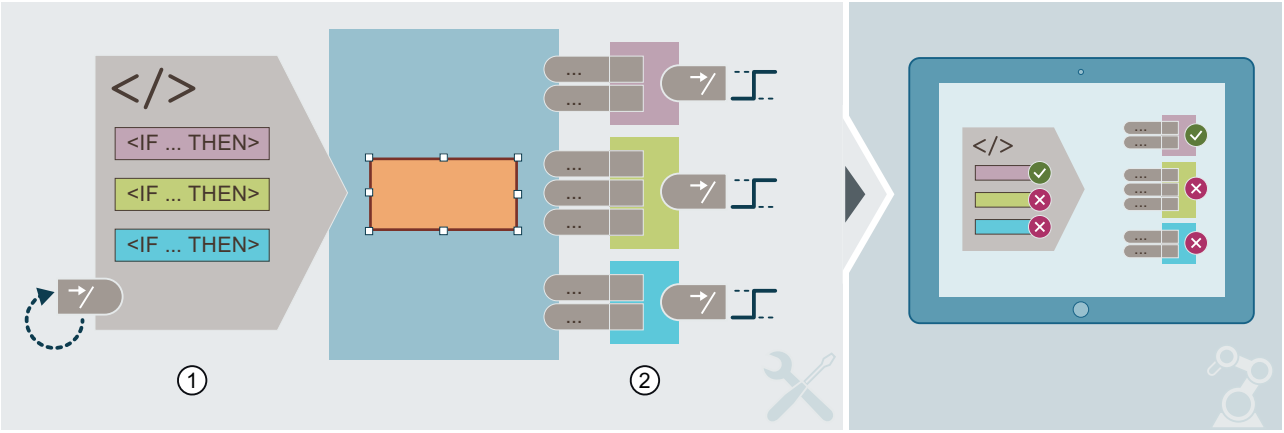
Global modules for frequently required functions

Global modules contain scripts which are available in the entire project. Global modules are therefore suitable for configuring frequently required functions.

Using scripts for dynamization

The properties of screens and screen items can be dynamized via local scripts. In addition, a screen or screen item has its own script area to create a "Global definition".

The "Global definition" is used to define modules, local tags and functions and to import other modules with "Import".



① Global definition for screen or screen item
② Local scripts per property

Input support

The "Scripts" editor assists you in entering code through:

- Syntax highlighting
- Snippets
- System functions
- Referencing HMI objects
- Tooltips
- Autocomplete
- Error marking and correction

See also

Global modules (Page 978)

Local scripts (Page 980)

Configuring a script to an event (Page 987)

Dynamizing object properties by script (Page 988)

Input support (Page 982)

10.1.2 Basics

Scripting environment

WinCC provides you with a modern scripting environment that you can use to automate a variety of system components, such as the graphical runtime system.

In the process, the scripting environment forms individual elements of the system components, such as the screens of the graphical runtime system, through an object model. You use this object model in your scripts to solve a variety of tasks or to control processes.

The script environment in WinCC offers:

- Efficiency and current technologies
  The scripting environment supports Unicode and uses JavaScript (JS) as the scripting language. The scripting environment is object-oriented and offers numerous asynchronous operations for high-performance and secure script execution.
- Support of mass data
  The script environment is optimized for the processing of mass data, for example the writing of 1000 tags in one pass. Special script objects are available to this purpose that record numerous HMI objects of the same type. These script objects execute operations on all the HMI objects simultaneously instead of processing each HMI object individually.

JavaScript

The script environment supports JavaScript according to the ECMAScript Language Specification. Google V8 is used as the script engine.

To find out which version of Node.js you are using, follow these steps:

1. Opens the file explorer.
2. Switch to "C:\Program Files\Siemens\Automation\WinCCUnified\WebRH\bin"
   The path may differ depending on the installation settings.
3. Open the shortcut menu of the file: "node.exe".
4. You can find the version used under "Details > Product version".

An overview of the functions that can be used in the individual versions of Node.js can be found under: https://node.green/ (https://node.green/).

Scripts in WinCC

As a rule the script environment executes all the scripts on the server side (NodeJS). Referenced external resources such as files used in the script therefore have to be available in the server environment.

Restrictions

The use of JavaScript is restricted when used with WinCC.

Functions that access external resources are not available:

- Access to the file system via the Node.js module "fs".
  Use WinCC Unified functions instead, for example `HMIRuntime.FileSystem.WriteFile()`, `HMIRuntime.FileSystem.ReadFile()`
- Connection to other web servers and access to content on the Internet, e.g. via `XMLHttpRequest` or `fetch()`.
- Import of other Node.js modules, e.g. via "import ... from ...".
  This applies to both Node.js modules provided by third-party vendors and Node.js modules provided with the current Node.js version.
  Only global modules of the runtime in TIA Portal can be imported.

To access other web servers and to import other Node.js modules, follow these steps:

- Create your own application.
  - On the Panel: Edge app
  - on the PC: any app, JavaScript is recommended.
- Run the application independently of WinCC Unified.
- Results of the application are passed to the runtime via OpenPipe and corresponding HTML tags.

10.1.3 Notes on creating scripts

Tips and tricks in SiePortal

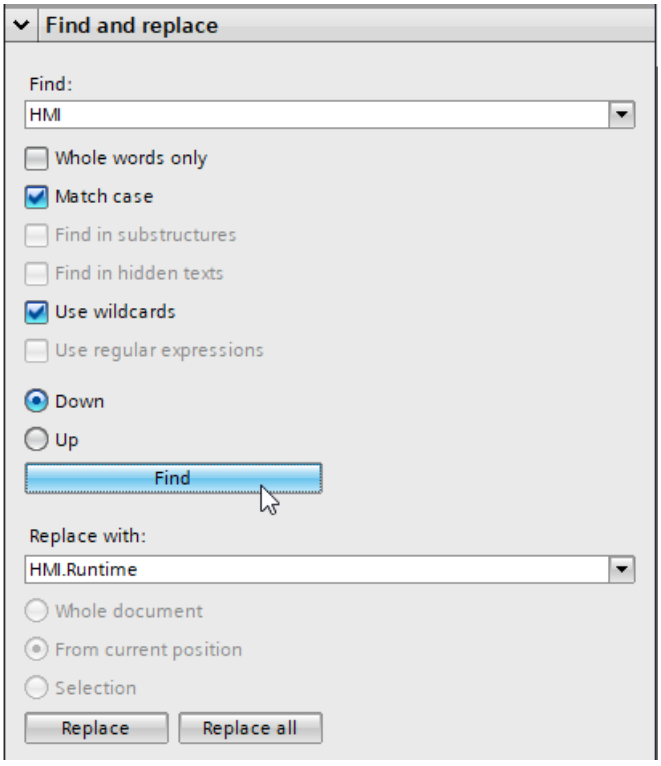
Get to know the options for creating scripts (JavaScript) in SIMATIC WinCC Unified and how you can quickly and easily create scripts with snippets. Tips and tricks on using objects via JavaScript are available for selected objects.

Tips and tricks in SiePortal: Tips and tricks for scripting (JavaScript) (https://support.industry.siemens.com/cs/ww/en/view/109758536)

Global search in scripts

Find and replace functions are available in a script.

1. To find strings that appear in a script or in the global definition range, use the global search in the project.
2. To open the "Find and replace" function, use menu "Edit > Search in project".
   - or -
   Press <Ctrl> + <F>.



The search is run on the script opened in the script editor.

The Find and replace function is also available if you open a script module type from a library for editing.

Floating-point numbers in JavaScript

JavaScript supports floating-point numbers with a mantissa of up to 54 bits. In scripting and Web client, values with a mantissa greater than 54 bits are therefore rounded.

This includes:

- Tag values
- Constants in the properties of screen objects

Note

Values with a mantissa of up to 64 bits are correctly displayed by I/O fields.

Triggering scripts

To improve performance, trigger the execution of scripts with tags. Avoid cyclic triggers.

Scripts with trigger tags

A script triggered by a tag is not allowed to write to this tag.

Using scripts during short cycle times

Calls of scripts with a short cycle time can lead to overloads.

Memory allocation by scripts

Memory usage by scripts is not limited in runtime. Pay particular attention to the size of the allocated memory when creating tags dynamically.

Application of graphic objects in runtime

Only those graphic objects which are referenced at least once, for example as an object reference in a script, are loaded in runtime. The assignment of a tag and the referencing of the tags does not function.

Example of correct referencing of a graphic object in a script:

```
Screen.Items("Grafikanzeige_1").Graphic =
HMIRuntime.Resources.Graphics("GraphicCollection.Up_Arrow").Name;
```

10.1.3.1 Data types

Data types in the object model

Unlike the basic JavaScript data types String, Number and Boolean, the data types in the WinCC object model are more typified.

The following table lists the utilized data types of the object model:

Data type	Description
Bool	Logical values (True/False)
UInt8	Unsigned 8-bit integer
Int8	Signed 8-bit integer
UInt16	Unsigned 16-bit integer
Int16	Signed 16-bit integer
UInt32	Unsigned 32-bit integer
Int32	Signed 32-bit integer
Float	Signed 32-bit floating-point number [1)]
String	Sequence of characters

Data type	Description
Variant	Object that can have any data type.
DateTime	Date/time information
StringStringMap	Map: Value pairs from strings
Promise	Object
Object	Object
Function	Method
ErrorCode	Error code

[1] JavaScript supports floating-point numbers with a mantissa up to 54 bits. In scripting and Web client, values with a mantissa greater than 54-bit are therefore rounded. This affects tag values and constants in the properties of screen objects.

Values with a mantissa of up to 64 bits are correctly displayed by I/O fields.

10.1.3.2 Object instances

Create objects

In the object model, all object instances are returned via access methods. There are no constructors that create objects.

Example

```
var t1 = Tags("Tag_1"); // returns HMITag object
var screenItem1 = Screen.FindItem("Button_1"); // returns HMIScreenItem
object
```

Error example

```
var t1 = new Tag("Tag_1"); // Error!
```

10.1.3.3 Enumerations

Description

Enumerations are enumeration types. Objects of this data type consist of elements with static values. Each permissible value has a unique name and an assigned integer.

Here is an example for the background fill pattern of screen objects. The enumeration has the name "HmiFillPattern" and defines the following 16 values with name and integer.

- Solid (0)
- Transparent (65536)
- Horizontal (131072)
- Vertical (131073)
- ForwardDiagonal (131074)
- BackwardDiagonal (131075)

- Cross (131076)
- DiagonalCross (131077)
- GradientHorizontal (1048576)
- GradientVertical (1048577)
- GradientForwardDiagonal (1048578)
- GradientBackwardDiagonal (1048579)
- GradientHorizontalTricolor (1048832)
- GradientVerticalTricolor (1048833)
- GradientForwardDiagonalTricolor (1048834)
- GradientBackwardDiagonalTricolor (1048835)

Use of enumerations

All enumeration can be found directly in context of their respective property.

Values of enumerations can be reference in scripts using the name or integer.

By way of example, the "GetSpecialFolder()" method of the "FileSystem" object is used for this in the following. This method can return the current temporary directory or the user directory according to the "FolderId" enumeration:

- TempDir (0): Directory for the temporary files
- HomeDir (1): Directory for the files of the current user

Referencing the value using the name of the element

Every enumeration can be addressed using the "Enums" object. This is followed by the name of the enumeration and the name of the desired element to reference the value:

Copy code

```
let tempDir =
HMIRuntime.FileSystem.GetSpecialFolder(HMIRuntime.FileSystem.Enums.FolderI
d.TempDir);
HMIRuntime.Trace("Temp folder path: " + tempDir);
```

Referencing the value using the integer of the element

The value can also be directly referenced using the integer of the element:

Copy code

```
let tempDir = HMIRuntime.FileSystem.GetSpecialFolder(0);
HMIRuntime.Trace("Temp folder path: " + tempDir);
```

10.1.3.4 Asynchronous operations

Promises in JavaScript

Promises are used in the script environment to handle asynchronous operations. A `Promise` object contains placeholders for results of an operation that are not yet known.

A `Promise` has the following status:

- Pending: Operation of the `Promise` object is still being executed.
- Settled: Operation of the `Promise` object has been completed.
  - Fulfilled: Operation was successful. Result is a value.
  - Rejected: Operation was unsuccessful. Result is a `reason`.
    `reason` may contain an error code for an object with text, links or any other conceivable contents of an object. For this reason, for targeted error processing, it is advisable for `reason` to use an instance of an `Error` object.

As soon as the operation has been completed, a corresponding Handler with the available result is called.

Promises minimize latencies (delay times caused by signal processing), because the script continues to be executed during evaluation. In contrast, a script stops with asynchronous calls. Promises allow clean error handling with the "`catch`" method.

Promises can be cascaded in order to transform results or to sequence asynchronous operations.

Using promises

In the simplest form, promises return a value or error which is processed with the "then" and "catch" methods of the `Promise` object:

```
getPromise()
.then(function(Value) {
    ...
})
.catch(function(ErrorCode) {
    ...
});
```

Cascaded promises each return a Promise and allow a sequence of asynchronous calls:

```
getPromise()
.then(return p1)
.then(return p2)
.then(return p3)
.catch();
```

The functions of the `Promise` objects are executed in the order p1 > p2 > p3 and the "catch" method is called in case of error. All tags of the calling function are also available in the internal functions.

10.1.3.5 Support for errors

The user has various options of diagnosing errors and then rectifying them.

Trace Log

WinCC writes a log file for every subsystem. This file contains helpful information of the respective subsystem for narrowing down possible error causes.

Note

The log files are located in the directory "%PROGRAMDATA%\Siemens\Automation\Logfiles\WinCC_Unified_SCADA_V*".

Trace Viewer

The Trace Viewer is an external application for the display and targeted filtering of Trace alarms.

See also

RTIL Trace Viewer (Page 1022)

10.1.3.6 Global modules

Global modules can only be linked with the global definition (script). All scripts except the global definition are simple functions.

The "import" statement must be a top-level statement.

Description

A global module is a container for one or more global functions with a shared definition area.

Several global modules can be created for each device.

Global modules are suitable for categorizing or grouping global functions.

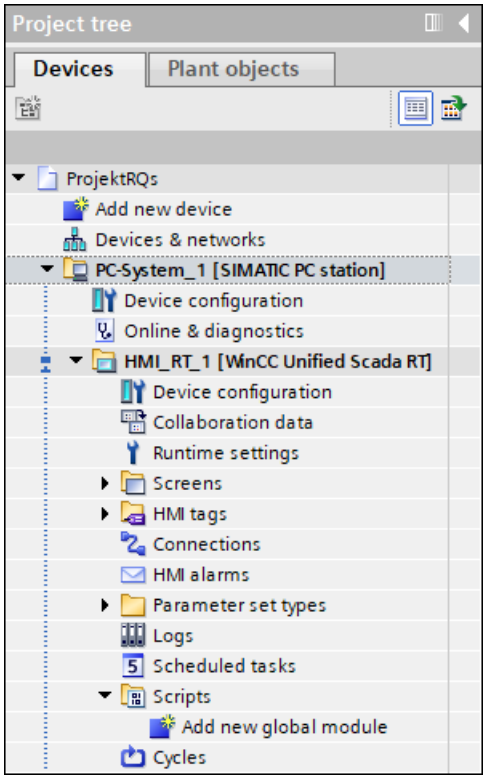
Note

Consider context when using global modules

After the import, a global module behaves like a local script. For example, it is not possible to use a global module on a task if a screen or screen object is referenced in the global module or an associated function. If an invalid reference to a screen or screen object is contained in a global module imported at a task, an error is output.

Call and view

Global modules are represented in the project tree in the respective device under "Scripts" as folder icons with the letters "JS" .



You can create new global modules via the shortcut menu of "Scripts".

You can create new global functions via the shortcut menu of the desired global module.

You can also add new global modules and functions via "Add ... new" in the project tree.

References to global modules

When you rename a global module, the alias is automatically renamed in all places where the global module is referenced.

References to functions defined in a global module are automatically updated in:

- Local scripts
- Global scripts
- Global definitions

Naming within the project remains consistent and runtime errors are avoided.

Examples

- General mathematical operations
- General logic operations
- Conversions of units of measurement

See also

Local scripts (Page 980)

10.1.3.7 Local scripts

A local script is a function written in JavaScript which is assigned to the object in question.

Starting local scripts

A local script is always started by a trigger:

- an event at a screen object
- a trigger on dynamization of object properties (cycle or change of a tag value)
- the event "Update" of a task in the Scheduler (cycle, change to a tag value or alarm)

Access to global modules

Local scripts can call functions which are contained in the scripts of global modules.

Applications

For example, local scripts can be used to

- dynamize object properties,
- process user entries and
- automatize complex operations

See also

Script and execution context (Page 985)
Configuring a script to an event (Page 987)
Dynamizing object properties by script (Page 988)
Creating a global definition in a local script (Page 988)
Basics of cycles (Page 223)

10.1.4 "Scripts" editor

10.1.4.1 Structure of the "Scripts" editor

In the "Scripts" editor, you create and edit customized JavaScript functions.

The "Scripts" editor can be opened in the following execution contexts:

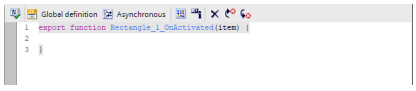
- Global functions in global modules
  You open the "Scripts" editor via the project tree by double-clicking a script.
- Local scripts which are triggered by events ("Images" editor and Scheduler)
  The "Scripts" editor is opened in the Inspector window under "Properties > Events" as soon as you have selected an event and selected the "Convert function list to script" button.
  Additional local scripts which are triggered by events can be triggered for each property under "Properties > Properties" for the events "Change" and "Quality code change".
- Local scripts for dynamizing object properties
  The "Scripts" editor is opened in the Inspector window under "Properties > Properties" as soon as you select the "Scripts" entry in the "Dynamization" column.

Different representation formats

Depending on the application, the "Scripts" editor is shown in different areas of TIA Portal and contains different operator controls.

Depending on the execution context, the code area opens either as a new editor window in the working area (global modules) or embedded in the Inspector window (local script).

Overview



The code area represents the actual JavaScript code.

Buttons are located above the code area. Depending on the application context, buttons are available with specific functions:

Button	Description	Global modules	Event-related functions	Dynamization of object properties
	Syntax check	✓	✓	✓
	Shows / hides the code area for the global definition.	×	✓	✓
	Specifies whether the function is synchronous/asynchronous	✓	✓	✓
	Enables the auto-completion of entries	✓	✓	✓
	Shows information or provides a tip for the place in the code where the insertion point is located	✓	✓	✓
	Deletes the script	×	✓	×
	Selects the trigger of dynamization (cycle or tag)	×	×	✓

Button	Description	Global modules	Event-related functions	Dynamization of object properties
	Sets the insertion point to the previous error in the code	✓	✓	✓
	Sets the insertion point to the next error in the code	✓	✓	✓

Note

Asynchronous functions cannot be used if the function returns a value.

Alternatively, the value can be specified via the respective property.

Shortcut menu

The shortcut menu contains so-called "snippets". Snippets provide frequently required code templates.

See also

Global modules (Page 978)

Configuring a script to an event (Page 987)

Dynamizing object properties by script (Page 988)

Input support (Page 982)

10.1.4.2 Input support

Introduction

You create the JavaScript code of your scripts in the code area of the "Scripts" editor.

The editor supports you with the following functions:

- Syntax highlighting
- Snippets (code templates)
- System functions
- Referencing HMI objects
- Tooltips
- Autocomplete
- Error marking and correction

Syntax highlighting

```
function OnAlarm(Errorcode, sysID, ResultSet) {
    HMIRuntime.Trace("Script OnAlarm Called");
    var index;
    var alarmCount = ResultSet.length;
    HMIRuntime.Trace("Alarm count = " + alarmCount);
    for (index = 0; index < alarmCount; ++index) {
        var name = ResultSet[index].Name;
        var alarm = HMIRuntime.Alarming.Alarms(name);
        alarm.CommentText = "Alarm is acknowledged and action done";
        alarm.ModificationTime = ResultSet[index].RaiseTime;
        alarm.Language = 1033;
        alarm.User = "My User Name";
        alarm.OperatorStation = "My Machine";
        var CommentErrorcode = alarm.SetComment();
        HMIRuntime.Trace("CommentErrorcode " + CommentErrorcode);
    }
}
```

The JavaScript code in the code area of the editor is highlighted in different colors to make it easier to read.

Snippets for programming support

Snippets are code templates for frequently required instructions: Snippets are divided into the following groups:

- HMIRuntime
  Contains Snippets for accessing the object model, e.g. "`Change base screen`" or "`Set Connection Mode`".
- Logic
  Contains Snippets such as "`If...Else`" or "`For loop`".
- Faceplate
  Only available in a faceplate.

You insert a Snippet in the local script via the shortcut menu.

System functions

The system functions are provided in the "Scripts" editor. For additional information, refer to "System functions".

Referencing HMI objects

HMI objects (e.g. tags and screens) are referenced in scripts.

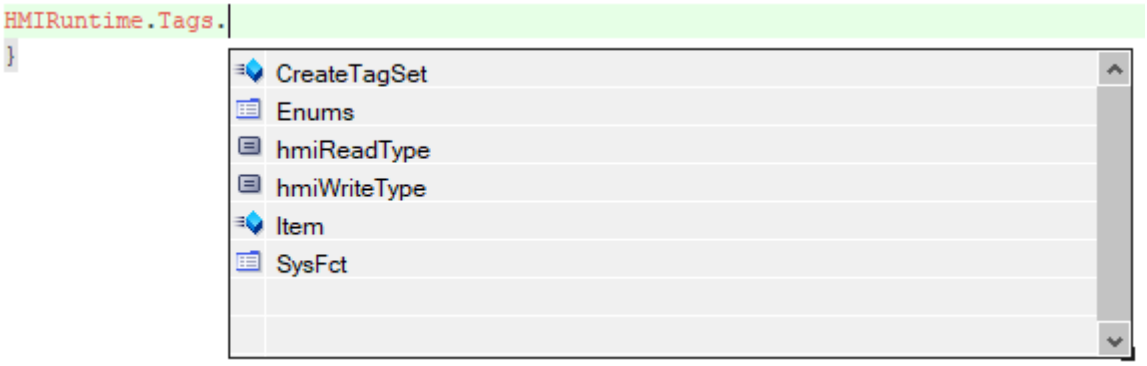
Therefore, you perform the following actions without editing the script:

- Rename HMI object
- Re-create a previously deleted HMI object
- Create HMI object used as text reference in the script

Info tips

While you compose the code, additional information about all objects of the WinCC Unified object model is displayed. For example, you receive information on the required parameters of the system functions.

Autocomplete



Autocomplete supports you in entering your code. Suitable objects of the WinCC Unified object model are displayed.

You call up autocomplete using the shortcut <Ctrl + I> or <Ctrl + Spacebar>.

<Ctrl+J>

The object selection can be called context-specifically by using the shortcut <Ctrl+J>. For example, you select screens, screen objects, tags, or graphics.

Error marking and correction

Errors can occur during compiling or during compiling and loading:

- Error while configuring
  Your JavaScript code is checked immediately during input for a variety of criteria and highlighted in color in case of errors.
  - To get a tooltip, move the cursor over the marking.
- Errors during compiling and loading
  Alarms during the compiling and loading of a project are displayed in the Inspector window in the "Info > Compile" tab.
  The "Scripts" editor supports you by displaying faulty scripts directly for editing:
  - To go directly to the "Scripts" editor, select the green arrow .

See also

System functions (Page 909)

10.1.4.3 Script and execution context

Introduction

Scripts are functions in the form of JavaScript code that you develop yourself.

You can use scripts to solve individual tasks, e.g.

- Automating processes
- Dynamizing objects
- Evaluating events, such as user input

The execution of a script is triggered by:

- An event
- A tag change
- A scheduled job

Preparation for creating and using scripts

In advance of creating the script, consider where you will enter the script and how it will be executed.

Depending on the objects and system components addressed, you enter the JavaScript code in different editors. The editors have a similar structure.

Depending on the execution context, the script is executed with different scopes.

Note

Each module has its own namespace, which means it is possible to use the same function name and global tag name in two modules.

The functions are distinguished by the symbol name of the imported module.

Example:

```
import * as modA from 'ModuleA';
import * as modB from 'ModuleB';
modA.function1();
modB.function1();
```

Execution context of the editor	Script context and referencing
"Script" editor in the "Screens" editor	Each screen has two independent script contexts: • Context for dynamization of properties • Context for evaluation of events The script of a property cannot access global tags of an event even in the same screen. Each script context references the modules that it has imported using the 'import' statement. However, each script context receives its own copy of the tags defined there.
"Script" editor for "Scheduled tasks"	All jobs share a script context. Different tasks can access common global tags. Each required module must be referenced by means of the 'import' statement.

Create "import" statement

A module can be referenced in a script or another module:

- Drag-and-drop the module to be referenced into the "Global Definition" area of the script or into the definition area of the module.

Note

Copying the "import" statement together with objects

If you copy a screen object and paste it into another screen, then the "import" statement is copied and pasted when the imported global module is used on the source object.

If the copied module has already been referenced under a different alias at the target, the copied module is inserted additionally.

See also

Simulating value changes in tags (Page 992)

Converting values (Page 997)

Local scripts (Page 980)

Configuring a script to an event (Page 987)

Dynamizing object properties by script (Page 988)

Creating a global definition in a local script (Page 988)

WinCC Unified object model (Page 1037)

10.1.4.4 Configuring a script to an event

Note

Restriction of "activated" and "deactivated" events

If the focus is on the affected screen item, scripts are executed at the "activated" and "deactivated" events.

Requirement

- One of the following objects is configured:
  - Task
  - Screen
  - Screen object

Procedure

To configure a script to an event, follow these steps:

1. Open the relevant editor.
2. Select the object.
3. Select the event under "Properties > Events" in the Inspector window.
4. Generate the local script.
5. Write the code.
6. Perform a syntax check.
7. Save the project.

See also

Introduction to runtime scripting (Page 969)

Local scripts (Page 980)

Script and execution context (Page 985)

Creating a global definition in a local script (Page 988)

10.1.4.5 Dynamizing object properties by script

Requirement

- One of the following objects is configured:
  - Screen
  - Screen object
- The selected object property supports the dynamization type "Script".

Procedure

1. Open the editor of the object in question.
2. Select the object.
3. Select the desired object property under "Properties > Properties" in the Inspector window.
4. Dynamize the object property:
   - Select the "Script" entry in the "Dynamization" column.
     The editor creates a script and is displayed in the Inspector window.
   - Write the code.
5. Select the trigger that triggers the dynamization in runtime.

Result

The script changes the selected property dynamically in line with the written code.

See also

Local scripts (Page 980)

Creating a global definition in a local script (Page 988)

10.1.4.6 Creating a global definition in a local script

Procedure

1. Open the local script.
2. Click "Global definition".
3. Write the code.
4. Perform a syntax check.
5. Edit the function.

See also

Local scripts (Page 980)

Script and execution context (Page 985)

Configuring a script to an event (Page 987)

Dynamizing object properties by script (Page 988)

10.1.5 Examples

10.1.5.1 Notes on the code examples

General

The comments at the beginning of each code example are required for technical reasons.

At the same time these comments show the relationships between various code examples within a chapter.

Further comments in the code examples explain individual program code lines.

Executing examples

1. Set the language of the develop environment to "English", so that the object names used in the examples are automatically assigned correctly.
2. Create a project with corresponding screens in which you can configure buttons, I/O fields, etc. These elements are required to carry out the code examples.
3. Apply the code examples to the associated script areas.
4. Compile the project.
5. Start the simulation of the project.
6. Start the tracer to diagnose potential errors.

See also

RTIL Trace Viewer (Page 1022)

10.1.5.2 Dynamizing the position of an object

Introduction

The example shows how to dynamically change an object position using JavaScript.

One classic application would be the adjustment of the object position to the size and/or position of adjacent objects on a screen.

Execution of example

1. Configure 4 tags with the names "HMI_Tag1" to "HMI_Tag_4".
2. Configure a screen with the following objects:
   - 4 objects of the type I/O field with the process values "HMI_Tag_1" to "HMI_Tag_4"
   - 1 object of the type "Circle" with the name "Circle_1"
   - 1 object of the type "Button" with the name "Button_1"
3. Dynamize the parameters "Position X" and ""Position" Y" of the objects "Circle_1" and "Button_1" using scripts.
4. Set the triggers for dynamization with the tags
   - HMI_Tag_1 for Button_1/Position X,
   - HMI_Tag_2 for Button_1/Position Y,
   - HMI_Tag_3 for Circle_1/Position X,
   - HMI_Tag_4 for Circle_1/Position Y.

   The scripts are started accordingly by changing the tag.
5. Copy the sample code below to your project.

Reading out and returning the tag **HMI_Tag_1**

The script with the function "`Circle_1_CenterX_Trigger(item)`" is created when the parameter "Position X" is dynamized by a script in the properties of the object "Circle_1".

```
//JEx: "Position X dynamization of a Circle with tags"
//TagsRequired: "HMI_Tag_1"
export function Circle_1_CenterX_Trigger(item) {
    var value = Tags("HMI_Tag_1").Read();
    return value;
}
```

Reading out and returning the tag **HMI_Tag_2**

The script with the function "`Circle_1_CenterY_Trigger(item)`" is generated when the parameter "Position Y" is dynamized by a script in the properties of the object "Circle_1".

```
//JEx: "Position Y dynamization of a Circle with tag"
//TagsRequired: "HMI_Tag_2"
export function Circle_1_CenterY_Trigger(item) {
    var value = Tags("HMI_Tag_2").Read();
    return value;
}
```

Reading out and returning the tag HMI_Tag_3

The script with the function "Button_1_Left_Trigger(item)" is generated when the parameter "Position X" is dynamized by a script in the properties of the object "Button_1".

```
//JEx: "Position X dynamization of a Button with tag"
//TagsRequired: "HMI_Tag_3"
export function Button_1_Left_Trigger(item) {
    var value = Tags("HMI_Tag_3").Read();
    return value;
}
```

Reading out and returning the tag HMI_Tag_4

The script with the function "Button_1_Top_Trigger(item)" is generated when the parameter "Position Y" is dynamized by a script in the properties of the object "Button_1".

```
//JEx: "Position Y dynamization of a Button with tag"
//TagsRequired: "HMI_Tag_4"
export function Button_1_Top_Trigger(item) {
    var value = Tags("HMI_Tag_4").Read();
    return value;
}
```

Result (in runtime)

The position of the object in question changes in accordance with the values entered in the I/O fields.

See also

Notes on the code examples (Page 989)

10.1.5.3 Reading and writing tag values

Introduction

The example shows how to read, multiply and write the values from two tags ("HMI_Tag_2", "HMI_Tag_3") to another tag ("HMI_Tag_1").

In practical use, the tags could represent the following parameters:

- "HMI_Tag_1": Apparent power S
- "HMI_Tag_2": Electrical voltage U
- "HMI_Tag_3": Electrical current I

Executing an example

1. Configure three tags "HMI_Tag_1" to "HMI_Tag_3" of the type "Int".
2. You configure the following objects on a screen:
   – 1 button (in the example "Button_1")
   – 3 I/O fields with the process values "HMI_Tag_1" to "HMI_Tag_3"
3. Create a script for the event "Click left mouse button".
   The JavaScript editor creates the function "`Button_1_OnTapped(item, x, y, modifiers, trigger)`".
4. Insert the example code.

Sample code

```
//JEx: "Reading and writing tag values"
//Tags_Required: "HMI_Tag_1"; "HMI_Tag_2"; "HMI_Tag_3"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    Tags("HMI_Tag_1").Write(Tags("HMI_Tag_2").Read() * Tags("HMI_Tag_3").Read());
}
```

See also

Notes on the code examples (Page 989)

10.1.5.4 Simulating value changes in tags

Introduction

This example shows how tags are supplied with values in defined time intervals by a simulation.

With the simulation, demo projects can be created or tested without process integration.

Note

Connection of existing process tags leads to influencing of processes

The simulation writes the calculated values to the tags without further test steps.

- Do not link any external tags; these remain linked to any existing process.
  The simulation thus influences the process in which the external tag is integrated.
- If you purposely want to influence a process with simulation, note that the external tag of the process can only be reached for simulation if the following requirements are met:
  – The connection to the controller (PLC) is established.
  – The controller (PLC) is in "RUN" mode.

The global functions for generating sine and sawtooth waves use 5 parameters:

- `counter`: Counter that uses the current date in milliseconds.
- `phase`: Phase offset as a factor between 0.0 and 1.0
  The factor 0.0 to 1.0 corresponds to a phase offset of 0° to 360°.
- `period`: Duration of a full vibration cycle in milliseconds
- `amp`: Strength of the amplitude
- `offset`: Shift of the amplitude on the y-axis

Executing an example

1. Create a global "Simulator" module.
2. Configure the two functions "sinWave" and "sawTooth" in this global module.
   Use the source codes from the following sections for the functions:
   - "Example code > Simulate sine wave (global script)"
   - "Example code > Simulate sawtooth wave (global script)"
3. Create a script for the "Loaded" event for the screen.
4. Go to the "Global definition" view of the event.
5. Insert the sample code under "Sample code > Event - Global definition area" in the "Global definition" view of the event.
6. If necessary, adapt the copied sample code to your project. For example, if you use more than 2 tags, you must add more lines with the function `ts.Add(...)`.
7. Go back to the "Function" view of the event.
8. Insert the source code from "Example code > Event".

The tags HMI_Tag_1 and HMI_Tag_2 can now be connected to any objects in the screen that can display the values, such as:

- Function trend control
- Gauges
- Bar graphs
- Text fields

Result

1. When the application is loaded in runtime, the `import` function initializes the scripts from the global module "Simulator" for the "Loaded" event of the screen.
   The example code can be found under "Event - Global definition area".
2. If the event is triggered when the screen is loaded, the function "simulateTags()" starts at the specified intervals.
   In the example, the interval is 500 ms.
   The call of the function "simulateTags()" is stopped as soon as the screen is deselected.
   The example code can be found under "Event" and "Event - Global definition area".
3. The function "simulateTags()" starts with each call of the functions "sinWave" and "sawTooth" and transfers the new values to the tags.
   The example code can be found under "Simulate sine wave (global script)" and "Simulate sawtooth wave (global script)".

Sample code

Simulate sine wave (global script)

```
//JEx: "Simulate Sine Wave"
export function sinWave(counter, phase, period, amp, offset) {
 return offset + amp * Math.sin((phase + ((counter % period) / period)) * (2*Math.PI));
}
```

Simulate sawtooth wave (global script)

```
//JEx: "Simulate Saw Tooth Wave"
export function sawTooth(counter, phase, period, amp, offset) {
 return offset + amp * (((counter + phase * period) % period) / period);
}
```

Global event definition area

```
//JEx: "Generate signals"
//SOM_OM_"HmiTrendControl"
//JExRequired: "Simulate Sine Wave", "Simulate Saw Tooth Wave"
import * as sim from "Simulator";
function simulateTags() {
 let counter = Date.now();
 let ts = Tags.CreateTagSet();
 ts.Add([['HMI_Tag_1', sim.sinWave(counter, 0.00, 10000, 25, 25)]]);
 ts.Add([['HMI_Tag_2', sim.sawTooth(counter, 0.25, 37000, 30, 15)]]);
 ts.WriteAsync();
}
```

Event

```
//JEx: "Event Screen_1 OnLoad"
//SOM_OM_"HmiTrendControl"
//JExRequired: "Generate signals"
export function Screen_1_OnLoad(item) {
 HMIRuntime.Timers.SetInterval(simulateTags, 500);
}
```

See also

Notes on the code examples (Page 989)

10.1.5.5 Using tag values globally

Introduction

The example shows how to use tag values globally. A tag value can therefore be shared between screens and tasks, for example.

The basic procedure is as follows:

1. The tag value is written into an internal tag via a script.
2. The tag value is read by another script at the desired place.

You can save and read the values of all data types supported by the object model in internal tags.

Executing an example

The following example writes a structured value of a JavaScript tag to an internal tag. Another member is added to the structured value in the sample code on button 2.

1. Configure an HMI tag "Tag".
2. Configure two buttons (in the example "Button_1" and "Button_2") on a screen.
3. Create a script for each "Click left mouse button" event of the buttons.
   The JavaScript editor creates the functions "`Button_1_OnTapped(item, x, y, modifiers, trigger)`" and `Button_2_OnTapped(item, x, y, modifiers, trigger)`.
4. Insert the sample code "Write structured value into internal tag" into the script of the first button.
5. Insert the sample code "Enhance structure by member "c"" into the script of the second button.
6. Compile and load it in runtime.
7. Trigger the event "Click left mouse button" on both buttons.

Result

Button 1

1. The JavaScript tag "tag" is created and initialized with the HMI tag "Tag".
2. The JavaScript tag "myObj" is created and initialized.
3. The JavaScript tag "myObj" is converted into a JSON string and assigned to the JavaScript tag "json".
4. The value of the JavaScript tag "json" is written into the HMI tag "Tag".

Button 2

1. The JavaScript tag "tag" is created and initialized with the HMI tag "Tag".
2. The JavaScript tag "myObj" is created and initialized.
3. The JavaScript tag "myObj" is extended by the member "c".
4. The JavaScript tag "myObj" is converted into a JSON string and assigned to the JavaScript tag "json".
5. The value of the JavaScript tag "json" is written into the HMI tag "Tag".

Sample code

Write structured value into internal tag

```
//JEx: "set initial values without 'c'"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
let tag = Tags('Tag');
let myObj = {a:10, b:20, pos: {x:100, y:200}, layers: [1, 8, 18, 24, 33]};
let json = JSON.stringify(myObj);
HMIRuntime.Trace('Jason ' + json);
tag.Write(json, 1);
}
```

Enhance structure by member "c"

```
//JEx: "add member 'c'"
export function Button_2_OnTapped(item, x, y, modifiers, trigger) {
const tag = Tags('Tag');
let myObj = JSON.parse(tag.Read(1));
myObj.c = (myObj.c || 0) + 1; // increment 'c'
let json = JSON.stringify(myObj);
HMIRuntime.Trace("New JSON: " + json);
tag.Write(json, 1);
}
```

10.1.5.6 Converting values

Introduction

The example shows how temperature values can be converted into a different unit with JavaScript.

Executing an example

1. Create a global module "TemperatureConversions" with 2 global scripts:
   - "celsiusToFahrenheit(t_celsius)"
   - "fahrenheitToCelsius(t_fahrenheit)"
2. Copy the corresponding sample code to the global scripts.
3. Copy the sample code from the "Global definition range" to the global definition range of the global module "TemperatureConversion".
4. Create a screen with 2 elements of the type "I/O field".
5. Dynamize the "Process value" property of the two "I/O field" elements using scripts.
6. Copy the sample code of both scripts.

Celsius to Fahrenheit (global script)

```
//JEx: "CelsiusToFahrenheit"
//JExRequired: "TempConv_GlobalDefRange"
export function celsiusToFahrenheit(t_celsius) {
    return t_celsius * 1.8 + 32;
}
```

Fahrenheit to Celsius (global script)

```
//JEx: "FahrenheitToCelsius"
//JExRequired: "TempConv_GlobalDefRange"
export function fahrenheitToCelsius(t_fahrenheit) {
    return (t_fahrenheit - 32) / 1.8;
}
```

Global definition range

```
//JEx: "TempConv_GlobalDefRange"
import * as tempConv from 'TemperatureConversion';
```

Celsius to Fahrenheit (dynamization of process value)

The JavaScript function is triggered by the tag 'celsius1'.

```
//JEx: "DynCelsiusToFahrenheit"
//SOM_OM_"HmiIOField"
//TagsRequired: "celsius1"
//JExRequired: "celsiusToFahrenheit"
//JExRequired: "TempConv_GlobalDef"
export function I_O_field_1_ProcessValue_Trigger(item) {
    const tagCelsius = Tags('celsius1');
    tagCelsius.Read();
    return tempConv.celsiusToFahrenheit(tagCelsius.Value);
}
```

Fahrenheit to Celsius (dynamization of process value)

The JavaScript function is triggered by the tag 'fahrenheit1'.

```
//JEx: "DynFahrenheitToCelsius"
//SOM_OM_"HmiIOField"
//TagsRequired: "fahrenheit1"
//JExRequired: "fahrenheitToCelsius"
//JExRequired: "TempConv_GlobalDef"
export function I_O_field_2_ProcessValue_Trigger(item) {
    const tagFahrenheit = Tags('fahrenheit1');
    tagFahrenheit.Read();
    return tempConv.fahrenheitToCelsius(tagFahrenheit.Value);
}
```

See also

Notes on the code examples (Page 989)

10.1.5.7 Change language

Introduction

The example shows how the interface language is changed by JavaScript.

Executing an example

1. Configure a button "Button_1".
2. Activate the project languages required for the example:
   - English
   - German
   - Hungarian
   - French
3. Create a script for the event "Left mouse button clicked" of the button "Button_1".
4. Define the tag "step" under "Global definition" and assign the value "2" to it.

Global definition

```
//JEx: "LanguageChangeGlobalDef"
var step = 2;
```

Sample code

Clicking the button increments the tag "step". The stored code for the language is assigned to `HMIRuntime.Language` according to its value.

```
//JEx: "LanguageChange"
//JExRequired: "LanguageChangeGlobalDef"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    //change to English
    if (step == 1) {HMIRuntime.Language = "1033";}
    //change to Hungarian
    if (step == 2) {HMIRuntime.Language = "1038";}
    //change to German
    if (step == 3) {HMIRuntime.Language = "1031";}
    //change to French
    if (step == 4) {HMIRuntime.Language = "1036";}
    step ++; //step
    if (step == 5) {step = 1;} //reset
}
```

See also

Notes on the code examples (Page 989)

10.1.5.8 Dynamically changing the output format of an object

Introduction

This example shows how to dynamically change the output format of an object of the type "I/O field" using JavaScript.

The output in the example is switched between the following formats:

- Hexadecimal
- Decimal
- Binary

Executing an example

1. Configure a screen page with 3 buttons with the names "Button_1" to "Button_3".
2. Configure max. nine objects of the "I/O field" type with the names "HmiIOField_1" to "HmiIOField_9".
3. Create a script for the event "Button pressed" for each button.
4. Define the constants as described in the section "Global definition of constants".
5. Create 2 global scripts:
   - toggleOutputFormat()
   - setOutputFormat(format)
6. Transfer the source code from the following sections.

Global script "toggleOutputFormat()"

```
//JEx: "Toggle Output Format"
//SOM_OM_"HmiIOField"
//JExRequired: "Set Output Format";"GlobalConstants for OutputFormat"
function toggleOutputFormat() {
    let index = outputFormats.indexOf(Screen.FindItem(screenItemNameBase +
minScreenItemIndex).OutputFormat);
    index = (index + 1) % outputFormats.length;
    setOutputFormat(outputFormats[index]);
}
```

Global script "setOutputFormat(format)"

```
//JEx: "Set Output Format"
//SOM_OM_"HmiIOField"
//JExRequired: "GlobalConstants for OutputFormat"
function setOutputFormat(format) {
    for(let i = minScreenItemIndex; i <= maxScreenItemIndex; i++) {
        let name = screenItemNameBase + i;
        Screen.FindItem(name).OutputFormat = format;
    }
}
```

Global definition of constants

```
//JEx: "GlobalConstants for SetOutputFormat"
const outputFormats = ['{I}', '{H2,2}', '{B8,4}'];
// Creating dynamic object names which differ by a number at the end
const screenItemNameBase = 'HmiIOField_';
// the screen item names begin with the prefix 'HmiIOField_'
const minScreenItemIndex = 1; // range of the screen items: min
const maxScreenItemIndex = 9; // range of the screen items: max
```

Switch to hexadecimal output format

```
//JEx: "Switch Output Format Hex"
//SOM_OM_"HmiIOField"
//JExRequired: "Set Output Format"
export function Button_1_OnDown(item, x, y, modifiers, trigger) {
    setOutputFormat('{H2,2}');
}
```

Switch to decimal output format

```
//JEx: "Switch Output Format Dec"
//SOM_OM_"HmiIOField"
//JExRequired: "Set Output Format"
export function Button_2_OnDown(item, x, y, modifiers, trigger) {
    setOutputFormat('{I}');
}
```

Switch to the next output format

```
//JEx: "Switch Output Format Bin"
//SOM_OM_"HmiIOField"
//JExRequired: "Toggle Output Format"
export function Button_3_OnDown(item, x, y, modifiers, trigger) {
    toggleOutputFormat();
}
```

See also

Notes on the code examples (Page 989)

10.1.5.9 Reading and writing binary files

Introduction

The example shows how to write a binary file to the file system with JavaScript.

Note

- For access to the data, use the class "DataView" or one of the "*Array" classes together with the class "arrayBuffer".
- If the Endianess of the computer systems used for development (Compiler) and execution (Runtime) are different, use the class "DataView".
- The method HMIRuntime.Trace(text) outputs whether the function was successful via the Tracer.

Executing an example

1. Configure 3 buttons with the names "Button_10", "Button_11" and "Button_14".
2. Create a script for the event "Print" for each button.
3. Copy the sample code below to your project.

Writing a binary file

```
//JEx: Write a binary file with class "Int32Array" into file system
export function Button_10_OnDown(item, x, y, modifiers, trigger) {
    var arrayBuffer = new Int32Array([1,2,3]);
    HMIRuntime.FileSystem.WriteFileBinary('C:\\Users\\Public\\binaryfile.bin',
arrayBuffer).then(
    function() {
        HMIRuntime.Trace('Write file finished successfully');
    }).catch(
    function(e) {
        HMIRuntime.Trace(`Write file failed with error code ${e}`);
    });
}
```

Reading a binary file with the class `Int32Array`

```
//JEx: Read a binary file with class "Int32Array" from file system
export function Button_11_OnDown(item, x, y, modifiers, trigger) {
    var arrayBuffer = new Int32Array([1,2,3]);
    HMIRuntime.FileSystem.ReadFileBinary('C:\\Users\\Public\\binaryfile.bin',
arrayBuffer).then(
    function(arrayBuffer) {
        let intView = new Int32Array(arrayBuffer);
        for(let i in intView) {
            HMIRuntime.Trace('intView[' + i  + '] = ' + intView[i]);
        }
    }).catch(
    function(e) {
        HMIRuntime.Trace(`Read file failed with error code ${e}`);
    });
}
```

Reading a binary file with the class `DataView`

```
//JEx: Read a binary file with class "DataView" from file system
export function Button_14_OnDown(item, x, y, modifiers, trigger) {
    HMIRuntime.FileSystem.ReadFileBinary('C:\\Users\\Public\\binaryfile.bin').then(
    function(arrayBuffer) {
        let dv = new DataView(arrayBuffer, 0, arrayBuffer.length);
        HMIRuntime.Trace('Int32[0] LE:' + dv.getInt32(0, true).toString(16));
        HMIRuntime.Trace('Int32[0] BE:' + dv.getInt32(0      ).toString(16));
        HMIRuntime.Trace('Int16[2] LE:' + dv.getInt16(2, true).toString(16));
    }).catch(
    function(e) {
        HMIRuntime.Trace(`Read file failed with error code ${e}`);
    });
}
```

See also

Notes on the code examples (Page 989)

10.1.5.10 Reading and writing text files

Introduction

This example shows how text files can be read and written with JavaScript.

Execution of example

Note

The method HMIRuntime.Trace(text) outputs whether the function was successful via the Tracer.

1. Configure 2 buttons "Button_12" and "Button_13".
2. Create a script for the event "Print" for each button.
3. Copy the sample code below to your project.

Writing sample code for text file

```
//JEx: "Write Text File"
export function Button_12_OnDown(item, x, y, modifiers, trigger) {
    HMIRuntime.FileSystem.WriteFile('C:\\Users\\Public\\textfile.txt', 'my utf8 string',
'utf8').then(
    function() {
        HMIRuntime.Trace('Write file finished successfully');
    }).catch(function(errorCode) {
        HMIRuntime.Trace('Write failed errorcode=' + errorCode);
    });
}
```

Reading sample code for text file

```
//JEx: "Read Text File"
export function Button_13_OnDown(item, x, y, modifiers, trigger) {
    HMIRuntime.FileSystem.ReadFile('C:\\Users\\Public\\textfile.txt',
'utf8').then(
    function(text) {
        HMIRuntime.Trace('Text=' + text);
    }).catch(function(errorCode) {
        HMIRuntime.Trace('Read failed errorcode=' + errorCode);
    });
}
```

See also

Notes on the code examples (Page 989)

10.1.5.11 Setting bits

Introduction

The example shows how single and multiple bits are masked and set with JavaScript.

Note

Using different methods for integer data types

The methods of the "`Math.Uint64`" object and the standard model are available for setting and resetting multiple bits.

The methods of the "`Math.Uint64`" object work with all integer data types.

- For values < 2^{31} use the standard methods of JavaScript.
- For values ≥ 2^{31} use `BigInt` of the standard model or the methods of the `Math.Uint64` object.

`BigInt` of the standard model is used in the following examples.

Executing an example

1. Create 6 buttons in a project.
2. Create a tag of the "Int" type.
3. For all 6 buttons create local scripts for the event "Left mouse button pressed".
4. Transfer the source code for the examples to the respective script areas.
5. To retain a clear overview assign descriptive texts to the buttons.

Setting a single bit (no error handling)

The "`Tag.SetBit()`" and "`Tag.ResetBit()`" methods are available for setting and resetting single bits.

```
//JEx: "SetSingleBit"
//SOM_OM_"
//TagsRequired: "HMI_Tag_1"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
    Tags('HMI_Tag_1').SetBit(37);
}
```

Setting a single bit

When error handling is included, a corresponding message is output with the `HMIRuntime.Trace()` method.

```
//JEx: "SetSingleBitWithErrorHandling"
//SOM_OM_"
//TagsRequired: "HMI_Tag_1"
export function Button_5_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const bitNum = 37;
    tag1.SetBit(bitNum).then(() => {HMIRuntime.Trace('SetBit succeeded');})
    .catch((e) => {HMIRuntime.Trace(`SetBit failed, error=${e}`);});
}
```

Changing bits with "Xor" without error handling

Changing multiple bits with 64 bits without error handling.

Copy code

```
//JEx: "SetMultipleBits"
//SOM_OM_"
//TagsRequired: "HMI_Tag_1"
export function Button_6_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');

    // Define a 64-bit mask using a binary constant
    const mask = 0b0110011000000000000000000000000000000001n;

    // Read LWord Tag
    tag1.Read();

    let newValue = BigInt(tag1.Value);
    newValue ^= mask; // use Bit-operations for clearing / setting bits

    tag1.Write(newValue);
}
```

Setting bits with "Or"

The function sets every bit in the mask that has the value "1".

The example uses asynchronous writing and includes an extended error handling.

```
//JEx: "SetMultipleBitsWithErrorHandlingOr"
//TagsRequired: "HMI_Tag_1"
// set bits with 'Or': sets every bit which is '1' in mask
export function Button_16_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const mask = 0b01100110000000000000000000000000000001n;

    tag1.Read();

    if(tag1.LastError != 0) {
        HMIRuntime.Trace(`Read failed, error=${tag1.LastError}`);
    } else if((tag1.QualityCode & 0x80) == 0) {
        // Check whether QC is 'good' or 'good cascade'
        HMIRuntime.Trace(
            `Read succeeded, but Quality is not 'good', QC=$
{tag1.QualityCode}`
        );
    } else {
        let newValue = BigInt(tag1.Value);
        newValue |= mask; // Set bits
        const ts = Tags.CreateTagSet([[tag1.Name, newValue]]);
        ts.WriteAsync().then(()=>{HMIRuntime.Trace('Write succeeded')})
        .catch((e)=>{HMIRuntime.Trace(`Write failed, error=${e}`)});
    }
}
```

Deleting bits with "AND"

The script deletes each masked bit with the value "1".

```
//JEx: "ClearMultipleBitsWithErrorHandlingAnd"
//TagsRequired: "HMI_Tag_1"
export function Button_17_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    let mask = 0b01100110000000000000000000000000000001n;

    // invert all bit of mask for 'And' operation
    mask ^= 0xfffffffffffffffn;
    tag1.Read();

    if(tag1.LastError != 0) {
        HMIRuntime.Trace(`Read failed, error=${tag1.LastError}`);
    } else if((tag1.QualityCode & 0x80) == 0) {
        // Check whether QC is 'good' or 'good cascade'
        HMIRuntime.Trace(
            `Read succeeded, but Quality is not 'good', QC=$
{tag1.QualityCode}`
        );
    } else {
        let newValue = BigInt(tag1.Value);
        newValue &= mask; // Delete bits

        const ts = Tags.CreateTagSet([[tag1.Name, newValue]]);
        ts.WriteAsync().then(()=>{HMIRuntime.Trace('Write succeeded')})
        .catch((e)=>{HMIRuntime.Trace(`Write failed, error=${e}`)});
    }
}
```

Replacing bits with "Xor"

The script replaces each bit in the mask that has the value "1".

```
//JEx: "FlipMultipleBitsWithErrorHandlingXor"
//TagsRequired: "HMI_Tag_1"
export function Button_12_OnTapped(item, x, y, modifiers, trigger) {
    const tag1 = Tags('HMI_Tag_1');
    const mask = 0b01100110000000000000000000000000000001n;

    tag1.Read();
    if(tag1.LastError != 0) {
        HMIRuntime.Trace(`Read failed, error=${tag1.LastError}`);
    } else if((tag1.QualityCode & 0x80) == 0) {
        // Check whether QC is 'good' or 'good cascade'
        HMIRuntime.Trace(
            `Read succeeded, but Quality is not 'good', QC=$
{tag1.QualityCode}`
        );
    } else {
        let newValue = BigInt(tag1.Value);
        newValue ^=mask;
        const ts = Tags.CreateTagSet([[tag1.Name, newValue]]);

        ts.WriteAsync().then(()=>{HMIRuntime.Trace('Write succeeded')})
        .catch((e)=>{HMIRuntime.Trace(`Write failed, error=${e}`)});
    }
}
```

See also

Notes on the code examples (Page 989)

10.1.5.12 Changing the date format

Introduction

The example shows how the date format is changed using JavaScript.

Executing an example

1. Create 2 I/O fields and 1 button.
2. Configure 2 global tags:
   - HMI_Tag_1 of the type "Int"
   - HMI_Tag_2 of the type "WString"
3. Create a script for the event "Press" of the button "Button_1".

Sample code

```
//JEx: "ChangeDateformat"
//SOM_OM_"HmiIOField_1", SOM_OM"HmiIOField_2"
//TagsRequired: "HMI_Tag_1:Int","HMI_Tag_2:WString"
export function Button_1_OnDown(item, x, y, modifiers, trigger) {
    //Create array with all month names
    var monthNames = [
        "January", "February", "March",
        "April", "May", "June", "July",
        "August", "September", "October",
        "November", "December"
    ];
    var date = new Date(); //Create tag with current date
    var day = date.getDate(); //Separation of the individual date components
    var monthIndex = date.getMonth();
    var year = date.getFullYear();
    var month = monthIndex + 1;
    //The "getMonth()" object contains 12 values from "0" to "11".
    //Set the date format
    switch (Tags("HMI_Tag_1").Read()) {
        case 1: Tags("HMI_Tag_2").Write(day +'/' +month +'/' +year); break;
       case 2: Tags("HMI_Tag_2").Write(year +'-'  +month+ '-' +day); break;
        case 3:
           Tags("HMI_Tag_2").Write(year +'-'  +monthNames[monthIndex] + '-'
+day);
            break;
        default:
            Tags("HMI_Tag_2").Write(day + ' ' + monthNames[monthIndex] + '
' + year);
            break;
    }
}
```

See also

Notes on the code examples (Page 989)

10.1.5.13 Monitoring alarms

Introduction

This example shows how to create and monitor active alarms.

The reason (NotificationReason) for which the alarms were sent can have the following values:

- `Unknown` (1)
- `Add` (1)
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring, for example "State = 1".

- `Modify` (2)
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarm will not result in notifications until the alarm meets the filter criteria again, such as "State = 1". In this case, "NotificationReason" is set to `Add`.

State-based and event-based monitoring

The respective client application determines whether or not notifications of alarms with the "NotificationReason" `Modify` or `Remove` are ignored.

For example:

- State-based monitoring: A customer wants to create a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: If an alarm is received, an email should be sent. Only the notification reason `Add` is relevant.

Execution of example

1. Configure a button (in the example "Button_1") on a screen.
2. Create a script for the event "Click left mouse button".
   The JavaScript editor creates the function "`Button_1_OnTapped(item, x, y, modifiers, trigger)`".
3. Insert the example code.
4. Compile and load it in Runtime.
5. Trigger the event "Click left mouse button" on the button.

Result

1. A customer begins monitoring with the filter criterion "State = 1".
2. An alarm is triggered. Runtime notifies the customer of the "NotificationReason" as follows:

NotificationReason	Description
Add	"State" is 1 The alarm has arrived.
Modify	"State" property has not changed. Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	"State" has changed. The alarm is removed.

Sample code

```
//JEx: "A client starts an alarm subscription with filter criterion "State = 1" (Raised
alarms)."
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
var subs = HMIRuntime.Alarming.CreateSubscription();
subs.Filter = 'State=1';
subs.Language = 1033;
subs.OnAlarm = function(Errorcode, SystemNames, ResultSet) {
  for (let index in ResultSet)
  {
    HMIRuntime.Trace('Alarm Name_' + (index+1) + ' = ' + ResultSet[index].Name);
    HMIRuntime.Trace(' Alarm State_' + (index+1) + '= ' + ResultSet[index].State);
    HMIRuntime.Trace(' Notification Reason_' + (index+1) + '= ' +
ResultSet[index].NotificationReason);
  }
}
subs.Start();
}
```

10.1.5.14 Set alarm filter

Introduction

The example shows how the alarm filter is set by a JavaScript function.

To make the examples easy, the filters are set by pressing a button.

Classic application example: Setting alarm filters using check boxes on a screen page.

Executing an example

1. Configure a screen with the following elements:
   - 1 alarm control ("Alarm control_1")
   - 2 buttons (in the example "Button_7" and "Button_8")
2. Configure the generation of alarms of the alarm class "Warning" and "Alarm".
3. Ensure that the alarms in the alarm control "Alarm control_1" are displayed.

Filtering sample code for "Warning"

In the sample code, alarms of the alarm class "Warning" are filtered.

```
//JEx: "Alarm Filter Control with Warnings"
//SOM_OM_"Alarm control" (Alarm control_1);
//JExRequired: "Alarm Subscription"
export function Button_7_OnDown(item, x, y, modifiers, trigger) {
    let alarmControl = Screen.FindItem('Alarm control_1');
    alarmControl.Filter = "AlarmClassName='Warning'";
}
```

Filtering sample code for "Warning" and "Alarm"

In the sample code, alarms of the alarm class "Warning" and "Alarm" are filtered.

```
//JEx: "Alarm Filter Control with Warnings and Alarms"
//SOM_ON_"Alarm control" (Alarm control_1);
//JExRequired: "Alarm Subscription"
export function Button_8_OnDown(item, x, y, modifiers, trigger) {
    let alarmControl = Screen.FindItem('Alarm control_1');
    alarmControl.Filter = "AlarmClassName IN ('Warning','Alarm')";
}
```

See also

Notes on the code examples (Page 989)

10.1.5.15 Creating an alarm subscription

Introduction

The example shows how an "Alarm subscription" is programmed in JavaScript.

Executing an example

1. Configure a task in the Scheduler with the time interval for the update.
2. Create a script for the event "Update" at the configured task.
3. Create the tag "subs" under "Global definition" (compare section "Global definition").
4. Copy the code under "Example code" to the script for the "Update" event.

Global definition

```
//JEx: "AlarmSubscriptionDef"
// A global tag is required for the Alarm Subscription,
// to be able to end the subscription
var subs;
```

Sample code

The JavaScript function in the example is started via the event "Update" of a task in the Scheduler.

```
//JEx: "AlarmSubsciption"
//JExRequired: "AlarmSubscriptionDef"
//TagsReqired: "HMI_Tag_1_Int"
// Function will be generated when you create a script on event "Update" of a task
export function Task_Task_1_Update() {
    let tag = Tags("HMI_Tag_1_Int");
    tag.Read();
    if(tag.Value == 1) {
        // start subscription
        if(subs) {
            // stop already existing subscription
            subs.Stop();
            subs =  undefined;
        }
        subs = HMIRuntime.Alarming.CreateSubscription();
        subs.Filter = 'AlarmClassName=\'Warning\'';
        //subs.Language = 1033; // For explicit setting of a specific language
        subs.OnAlarm = function(Errorcode, SystemNames, ResultSet) {
            // Callback function is performed as soon as the alarm attribute changes
            for (let index in ResultSet) {
              HMIRuntime.Trace('Alarm[' + index + '] Name="' + ResultSet[index].Name + '"');
                HMIRuntime.Trace('Alarm[' + index + '] State="' + ResultSet[index].State +
'"');
                 HMIRuntime.Trace('Alarm[' + index + '] EventText="' +
ResultSet[index].EventText + '"');
            }
        }
        subs.Start();
    } else if(tag.Value == 2) {
        // stop subscription
        if(subs) {
            // Stop already existing Alarm Subscription
            subs.Stop();
            subs = undefined;
        }
    }
}
```

See also

Notes on the code examples (Page 989)

10.1.5.16 Creating alarms with multilingual alarm texts

Introduction

The example shows how you transfer multilingual alarm text to the system function "CreateSystemAlarm" using text lists.

In the first sample code, you are using a text list entry in the "alarmText" parameter. In this entry, you refer to the "alarmParameterValue1" parameter.

In the second sample code, you are also using a text list entry in the "alarmText" parameter. In this entry, you refer to an additional text list.

This example can be transferred to the system functions "CreateSystemInformation" and "CreateOperatorInputInformation".

The system functions "CreateSystemInformation", "CreateSystemAlarm" and "CreateOperatorInputInformation" have the following parameters:

- "alarmText": Specifies the alarm text.
  You have the option of establishing references to the parameter values.
- "area" (optional): Specifies the scope.
- "alarmParameterValue1" to "alarmParameterValue7" (optional): Specifies the alarm parameters.

Tips for an efficient procedure

You can find the following snippets for the system function "CreateSystemInformation" in the shortcut menu of the "Scripts" editor under "Snippets > HMI Runtime > Alarming":

- "CreateSystemInformation with monolingual Alarm Text"
- "CreateSystemInformation with multilingual Alarm Text and Parameter Value"
- "CreateSystemInformation with multilingual Alarm Text and embedded Text List"

Executing an example

1. Create multiple project languages. One of the languages is English.
2. Activate the desired project languages in the runtime settings of the HMI device under "Language & Font".
3. Create a text list "Text_list_3" of the type "Value/Area".
4. Create an entry in the text list.
5. Set the value to "1".
6. Specify the English text: "Parameter value 1: @1%s@".
   "Parameter value 1 :" can be displayed in multiple languages.
   "@1%s@" is the reference to the "alarmParameterValue1" parameter. "1" indicates the number and "s" the data type. The data type is String.
7. Specify the text for additional languages in the Inspector window under "Properties > Texts".
8. Create a text list "Text_list_1" of the type "Value/Area".
9. Create an entry in the text list.

10. Set the value to "1".
11. Specify the text for all required languages: "@1%t#2T@".
    "@1%t#2T@" is the reference to the parameters "alarmParameterValue1" and "alarmParameterValue2".
    The name of the text list in transferred in "alarmParameterValue2".
    The value of the text list entry is transferred in "alarmParameterValue1".
12. Create a text list "Text_list_2" of the type "Value/Area".
13. Create an entry in the text list.
14. Set the value to "1".
15. Specify the English text: "MyText".
16. Specify the text for additional languages in the Inspector window under "Properties > Texts".
17. Configuring the buttons "Button_1" and "Button_2".
18. Configure an alarm control.
19. Configure the alarm control so that current alarms of the alarm class "SystemAlarm" are displayed.
20. Create a script for the event "Click left mouse button" at each of the buttons.
21. Add the respective sample code.
22. Configure buttons for switching the language.
23. Compile and load the project.
24. Trigger the events at the buttons.

Result

"Button_1"

- When the runtime language "English" is active, the alarm of the alarm class "SystemAlarm" is displayed as follows in the alarm control:
  - The area is "MyArea".
  - The alarm text is "Parameter value 1: My Parameter Value 1".
    The alarm text comes from the entry of the text list "Text_list_3".
- When you switch the language in runtime, the language-dependent alarm text is displayed. "My Parameter Value 1" is language-neutral.

"Button_2"

- When the runtime language "English" is active, the alarm of the alarm class "SystemAlarm" is displayed as follows in the alarm control:
  - The area is "MyArea".
  - The alarm text is "MyText".
    The alarm text comes from the entry of the text list "Text_list_2".
- When you change the language in runtime, the language-dependent message is displayed.

Sample codes

"Button_1"

```
//JEx: "CreateSystemAlarm with multilingual Alarm Text and Parameter Value"
//TextListsRequired: "Text_list_3"
//EntryRequired: value: "1", Text: "Parameter value 1: @1%s@"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
let value = 1;
HMIRuntime.Alarming.SysFct.CreateSystemAlarm(
  HMIRuntime.Resources.TextLists("Text_list_3").Item(value), "MyArea", "My
Parameter Value 1");
}
```

"Button_2"

```
//JEx: "CreateSystemAlarm with multilingual Alarm Text and Parameter Value"
//TextListsRequired: "Text_list_1", "Text_list_2"
//Text_list_1: value: "1", Text: "@1%t#2T@"
//Text_list_2: value: "1", Text: "MyText"
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
let value = 1;
HMIRuntime.Alarming.SysFct.CreateSystemAlarm(
 HMIRuntime.Resources.TextLists("Text_list_1").Item(value),
 "MyArea", 1, HMIRuntime.Resources.TextLists("Text_list_2"));
}
```

See also

CreateOperatorInputInformation (Page 915)

CreateSystemAlarm (Page 917)

CreateSystemInformation (Page 916)

10.1.5.17 Opening and closing a screen in a pop-up window

Introduction

The example shows you how to open and close a screen in a pop-up window. To do this, you use the system functions "OpenScreenInPopup" and "ClosePopup".

OpenScreenInPopup

The system function "OpenScreenInPopup" has the following parameters:

- "popupWindowName": Specifies the name of the pop-up window.
  The name must be unique within the parent screen.
- "screenName": Specifies the name of the screen that is opened in the pop-up window.
- "toggleOpen": Indicates whether the pop-up window is closed when the function is called again.
- "Caption": Specifies the title of the pop-up window.
- "Left": Defines the window position as offset from the left margin.
- "Top": Defines the window position as offset from the top margin.
- "hideCloseButton": Specifies whether to display the "Close" button.
- "parentScreenPath" (optional): Path of the parent screen. With this parameter you specify whether the popup window closes when a screen change is executed in the screen or screen window.
  You can specify the path absolute or relative:
  - Absolute path of the parent screen: "~/Screen"
  - Relative path of the parent screen: "./Screen"
  - Absolute path of the parent screen in a screen window: e.g. "~/Screen window_1/Screen"
    "Screen window_1" is the name of the screen window.
  - Relative path of the parent screen in a screen window: e.g. "./Screen window_1/Screen"
    "Screen window_1" is the name of the screen window.

  Relative path information is used in the following sample codes.

ClosePopup

The system function "ClosePopup" has the parameter "popupWindowPath". The parameter specifies the path of the pop-up window to be closed.

The "popupWindowPath" value is the result of the parameters "popupWindowName" and "parentScreenPath" of the system function "OpenScreenInPopup".

The parameter "popupWindowPath" is specified accordingly as relative or absolute path:

- Absolute path of the popup window: e.g. "~/MyPopup"
  "MyPopup" is the name of the popup window.
- Relative path of the popup window: e.g. "./MyPopup"
  "MyPopup" is the name of the popup window.
- Absolute path of the popup window with screen window: "~/Screen window_1/MyPopup"
  "MyPopup" is the name of the popup window and "Screen window_1" is the name of the screen window.
- Relative path of the popup windows with screen window: "./Screen window_1/MyPopup"
  "MyPopup" is the name of the popup window and "Screen window_1" is the name of the screen window.

Note

When the "parentScreenPath" parameter of the system function "OpenScreenInPopup" is not defined, the parameter is "popupWindowPath", for example, "/MyPopup". "MyPopup" is the name of the popup window.

Executing an example

1. Configure five screens: "Screen_1", "Screen_2", "Screen_3", "Screen_4" and "Screen_5".
2. Configure the following objects in the screen "Screen_1":
   - Five buttons: "Button_1", "Button_2", "Button_3", "Button_4" and "Button_5"
   - One screen window "Screen window_1"
3. Configure a screen change to "Screen_5" at the button "Button_5".
4. Add a button in "Screen_5" that triggers a screen change to "Screen_1".
5. Add a button in "Screen_3" that triggers a screen change to "Screen_4".
6. Add a button in "Screen_4" that triggers a screen change to "Screen_3".
7. Select "Screen_3" for the "Screen" property of the screen window.
8. Create a script for the event "Click left mouse button" at the buttons "Button_1", "Button_2", "Button_3" and "Button_4".
9. Add the respective sample code.
10. Compile and load the project.
11. Trigger the events at the buttons.

Result

- "Button_1":
  - "Screen_2" is opened in the pop-up window.
  - The pop-up window is closed when you change the screen by clicking "Button_5".
- "Button_2":
  - The pop-up window that was opened with "Button_1" is closed.
- "Button_3":
  - "Screen_2" is opened in the pop-up window.
  - When you change the screen in the screen window, the pop-up window is closed.
- "Button_4":
  - The pop-up window that was opened with "Button_3" is closed.

Sample code "Button_1"

```
//JEx: "OpenScreenInPopup with linked parentScreen"
export async function Button_1_OnTapped(item, x, y, modifiers, trigger) {
HMIRuntime.UI.SysFct.OpenScreenInPopup("MyPopup", "Screen_2", true,
"Popup", 100, 100, false, "./Screen");
}
```

Sample code "Button_2"

```
//JEx: "ClosePopup with linked parent screen"
export async function Button_2_OnTapped(item, x, y, modifiers, trigger) {
HMIRuntime.UI.SysFct.ClosePopup("./MyPopup");
}
```

Sample code "Button_3"

```
//JEx: "OpenScreenInPopup with linked parent screen in screen window"
//SOM_OM_"hmiScreenWindowInterface"
export async function Button_3_OnTapped(item, x, y, modifiers, trigger) {
HMIRuntime.UI.SysFct.OpenScreenInPopup("MyPopup", "Screen_2", true,
"Popup", 100, 100, false, "./Screen window_1/Screen");
}
```

Sample code "Button_4"

```
//JEx: "ClosePopup with linked parent screen in screen window"
//SOM_OM_"hmiScreenWindowInterface"
export async function Button_4_OnTapped(item, x, y, modifiers, trigger) {
HMIRuntime.UI.SysFct.ClosePopup("./Screen window_1/MyPopup");
}
```

See also

ClosePopup (Page 941)

OpenScreenInPopup (Page 935)

10.1.5.18 Triggering a screen change with a tag

Introduction

The example shows how you can trigger a screen change in a screen window using a tag. In the example, the tag value is changed by clicking on the buttons.

Application example: The system contains a motor that is controlled with a PLC. As soon as the motor displays a fault, a detailed view of the motor is displayed on the HMI device.

Executing an example

1. Configure three screens: "Screen_Motor", "Screen_Error" and "Screen_NoError".
2. Define different background colors for the screens "Screen_Error" and "Screen_NoError".
3. Create the tag "PLC_State_Tag" of the data type "Int".
4. Configure the following objects in the screen "Screen_Motor":
   - Two buttons, "Error" and "NoError"
   - One screen window
5. At the "Error" button configure the system function "SetTagValue" for the event "Click left mouse button":
   - Tag: "PLC_State_Tag"
   - Value: "1" (data type "Int")
6. At the "NoError" button configure the system function "SetTagValue" for the event "Click left mouse button":
   - Tag: "PLC_State_Tag"
   - Value: "0" (data type "Int")
7. Select the screen window.
8. Under "Properties > Properties > General" in the Inspector window, select the "Screen" property.
9. In the "Dynamization" column, select the "Script" option.
10. Add a trigger.
11. Select "PLC_State_Tag" as the trigger tag.
12. Insert the sample code.
13. Compile and load the project.
14. Trigger the events at the buttons.

Result

- The screen window shows the screen "Screen_Error" by default.
- When you click the "Error" button, the screen in the screen window changes to "Screen_Error".
- When you click the "NoError" button, the screen in the screen window changes to "Screen_NoError".

Sample code

```
//JEx: "ActivateScreenByNumber"
//TagsRequired: "PLC_State_Tag"
//SOM_OM_"HMIScreenWindowInterface"
export function Screen_window_1_Screen_Trigger(item) {
    let screenName;
    // Read the state tag first which is the same as the trigger tag
    let tag1 = Tags("PLC_State_Tag");
    let value1 = tag1.Read();
    // Convert state to screen name
    switch(value1)
    {
      case 0: screenName = "Screen_NoError";
      break;
      case 1: screenName = "Screen_Error";
      break;
      default: screenName = "Screen_NoError";
      break;
    }
    return screenName;
}
```

10.1.6 Troubleshooting

10.1.6.1 RTIL Trace Viewer

Core statement

The RTIL Trace Viewer is a separate application which runs independently of the TIA Portal, but which can be integrated into the TIA Portal as an "external application".

Principle

During runtime, the RTIL Trace Viewer displays all alarms which are listed in the configurable TraceCatalog.

Layout

The traces are displayed in tabular form and can be sorted in ascending and descending order by the columns displayed.

Filters

The required alarms can be filtered using filters. Alarms in non-selected categories are hidden.

File functions

You export alarms as trace logs in the following formats:

- Text file (.txt, .log): Loading and evaluation in the RTIL Trace Viewer
- .csv file: Evaluation in conventional spreadsheet programs or other csv-compatible applications

See also

Support for errors (Page 978)

10.1.6.2 Integrate RTIL Trace Viewer as an external application

Procedure

1. Open the settings via "Tools > Settings".
2. Open the category "External applications".
3. Double-click in the first empty line.
   An input mask for the external application opens.
4. Assign a descriptive name for the application, e.g. "RTIL Trace Viewer" in the field "Name".
5. Insert the following path under "Command": `%ProgramFiles%\Siemens\Automation\WinCCUnified\bin\RTILtraceViewer.exe`
6. Leave the fields "Arguments" and "Start in" empty.
7. Click "Add" and then close the "Settings" dialog.

The application is now available via the menu "Tools > External applications".

10.1.6.3 Tracing with the RTIL Trace Viewer

There is a "Trace Viewer" for finding errors in the JavaScript code.

Preparation

The "Trace Viewer" are located in the following path:

- %ProgramFiles%\Siemens\Automation\WinCCUnified\bin\RTILtraceViewer.exe

Requirement

- Simulation or runtime are started.
- RTILtraceViewer.exe has been started.

Procedure

1. Carry out an action in the simulation which starts a JavaScript function.
2. Filter by the trace messages "Subsystem > ScriptFW".

   Note

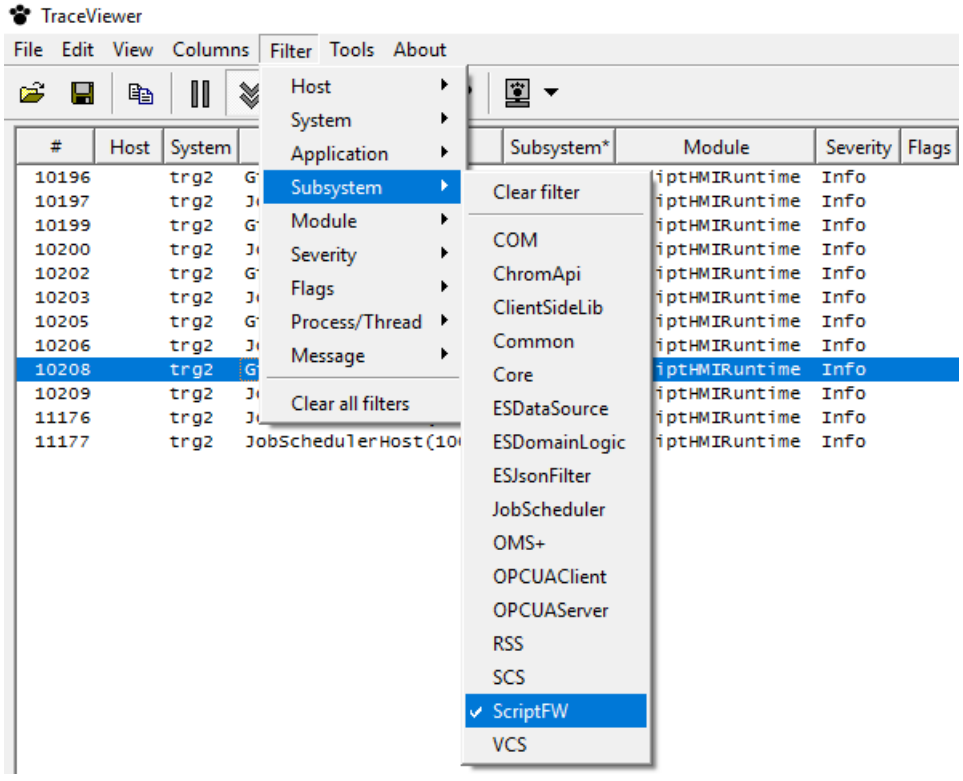
   As long as no JavaScript function was executed in the simulation during the runtime of the Trace Viewer, the entry "ScriptFW" is missing in the "Filter > Subsystem" menu.

3. Define additional filter criteria if required.
4. Evaluate trace messages if actions in the simulation lead to errors.

   Note

   If no messages are displayed in the Trace Viewer despite errors in the simulation, reset the filters:

   - Only in the submenu, e.g. "Subsystem": "Filter > Subsystem > Clear filter" menu
   - All filters: "Filter > Clear all filters" menu



See also

Integrate RTIL Trace Viewer as an external application (Page 1023)

10.1.7 Debugging scripts

10.1.7.1 Basics of debugging

Introduction

For example, you can use a debugger to test whether correct values are being transferred to tags and whether abort conditions are being correctly implemented. Check the following in the debugger:

- Source code of functions
- Function sequence
- Values

Note

Your code is displayed in the debugger but is write-protected.

Basic procedure

To find an error, check the script with the debugger.

The following options are available for your support:

- Setting breakpoints
- Step-by-step execution
- Viewing values parallel to execution of the script

You do not edit the code of your scripts directly in the debugger. When you find an error, follow these steps:

1. Correct the error in the engineering system.
2. Compile the changed code.
3. Load the runtime.
4. Update the debugger.

See also

Starting the debugger (Page 1029)

10.1.7.2 Design and function of the debugger

Google Chrome provides the user interface of the debugger. Not all functions of the user interface of the debugger are relevant for debugging WinCC Unified Scripts. Only the functions that are needed to debug scripts in WinCC Unified are explained below.

You can find more information on Chrome DevTools under: https://developers.google.com/web/tools/chrome-devtools/ (https://developers.google.com/web/tools/chrome-devtools/).

The debugger is divided into two areas:

- Debugger for screens
- Debugger for jobs

With the debugger for screens you view scripts at screens and screen objects. With the debugger for jobs, you view scripts that you have configured in the Scheduler.

Start page of the debugger

After the debugger has been started, its start page is displayed.

The available contents differ depending on the selected area.

On the start page of the debugger for screens you can see two different contexts:

- Dynamizations (e.g. "UMCadmin@192.168.116.144 VCS_1 Dynamics")
- Events (e.g. "UMCadmin@192.168.116.144 VCS_1 Events")

The name of the contexts is composed as follows:

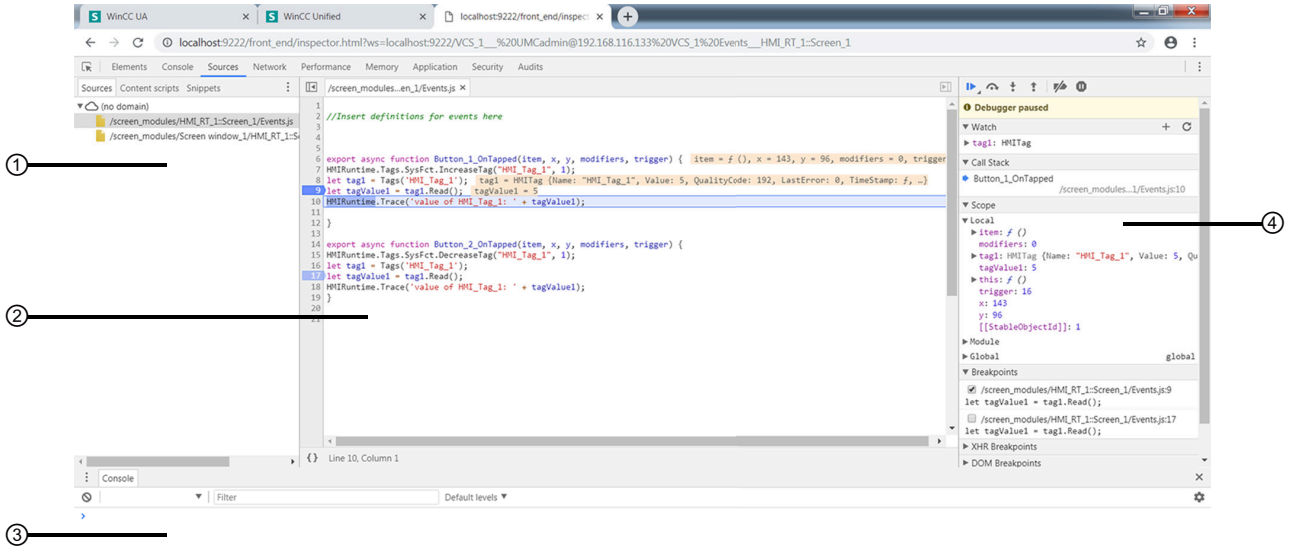
- UMCadmin: User name
- 192.168.116.144: IP address of the computer
- VCS: Name of the graphic component
- _1: Number of the open client
- Events/Dynamics: Scripts at events or dynamizations

Note

A client corresponds to a tab in Google Chrome in which the runtime is open. When you have opened runtime in multiple tabs, multiple clients are used. The client opened first is given the number 1. Numbering is reset when the runtime is restarted.

On the start page of the debugger for jobs you can see the context "JobsExecution".

User interface of the debugger



① Navigation area
② Code display area
③ Console
④ Debugging area

Navigation area

In the navigation area, the available contents for the screen shown in runtime are displayed in groups. The available groups vary depending on the use of scripts and functions.

Groups in the debugger for screens

The debugger for screens can contain the following groups in the dynamizations context:

- A group for scripts that were configured for dynamizations.
- One group per screen window in which scripts were configured for dynamizations.

The debugger for screens can contain the following groups in the events context:

- A group for scripts that were configured for events.
- One group for functions that were configured for events using the function list.
- One group per screen window in which scripts were configured for events.
- One group per screen window in which functions were configured for events using the function list.

Groups in the debugger for jobs

The debugger for jobs can contain the following groups:

- A group for scripts that were configured for tasks.
- One group for functions that were configured for tasks using the function list.

Code display area

Your code is displayed in the code display area. The rows are numbered.

Debugging area

The debugging area offers the following relevant options for WinCC Unified:

- Toolbar: Control for executing the script
- "Watch": Display of values
- "Callstack": Display of the current call stack
- "Scope": Available local values ("Local"), functions ("Module") and global values ("Global"),
- "Breakpoints": List of set breakpoints

10.1.7.3 Enabling the debugger

Requirement

- SIMATIC Runtime Manager is installed.
- The logged-on user has administrator rights.

Note

The debugger is only available locally.

Remote access from the debugger to other devices is not possible.

Procedure

The debugger is disabled by default.

Note

The debugger should be deactivated in production operation, as using the debugger can endanger system stability and security. Actions can accumulate if the debugger is, for example, at a breakpoint for a long time or the screen is not refreshed.

To activate the debugger, follow these steps:

1. Run the SIMATIC Runtime application with administrator rights.
2. Click the button  in the toolbar.
3. Switch to the "Scripts Debugger" tab.
4. To enable the debugger for screens, select the "Enable" check box in the "Screen debugger" area.
5. To enable the debugger for scheduled tasks, select the "Enable" check box in the "Scheduler debugger" area.
6. Assign an available port number to the debugger for screens (default port number: 9222).
7. Assign an available port number to the debugger for jobs (default port number: 9224).
8. Confirm your entries.

Note

Start the runtime after enabling the debugger.

10.1.7.4 Starting the debugger

Requirement

- Google Chrome (as of version 70) is installed.
- A project is opened in runtime.
- The debugger was activated in SIMATIC Runtime Manager.

Note

The debugger is only available locally.

Remote access from the debugger to other devices is not possible.

Procedure

1. In a new tab, call up the URL chrome://inspect in Google Chrome.
2. The home page of the Chrome DevTools is loaded in the tab.
3. Click "Devices".
4. Select the "Discover network targets" check box.
5. Click "Configure".

6. In the "Target discovery settings" dialog box, enter one of the following strings:
   - `127.0.0.1:<Port number>`
   - `localhost:<Port number>`

   Use the port entered for the Script Debugger in SIMATIC Runtime Manager.
7. Press <Enter>.
8. Click "Done".
9. Under "Remote Target", click "inspect" for the desired target.
   The DevTools open in a separate window with the selected target.
10. In the DevTools, select "Sources".
    The debugger is displayed.
11. Click "Toggle screencast".
12. In the navigation area under "Page", select the desired script module.

Updating the debugger

The debugger must be updated:

- After starting a new project
- After restarting a running project, for example, because you have reloaded the project in engineering with "Download to device > Software (all)".
- After a screen change in Runtime

The connection to the debugger is lost in each case. Google Chrome therefore shows an error message and asks whether you want to restore the connection.

To restore the connection, proceed as follows:

1. Close the DevTools window.
2. On the DevTools start page under "Remote Target", click "inspect" again for the desired target.

Stopping the debugger

Exit the debugger by closing the DevTools window and, if necessary, the DevTools homepage.

This does not stop runtime.

See also

Design and function of the debugger (Page 1025)

10.1.7.5 Working with breakpoints

Set breakpoints to stop the execution of the script at certain points and thus localize errors step-by-step. Previously set breakpoints are still available after updating the debugger.

Requirement

- Runtime has started.
- The debugger has been started.
- The group you want to debug is selected.

Pause script

To pause the execution of a script, you have 2 options:

- To pause the script immediately, click the "Pause script execution" button while the script is being executed.
- Set a breakpoint in the desired line.
  The script pauses when a breakpoint is reached.

To pause a script at a breakpoint that is configured to an event, follow these steps:

1. Set a breakpoint in the script.
2. Trip the respective event in runtime.
   The script pauses at the breakpoint.

Setting breakpoints

You have several options to set a breakpoint in a line of the script:

- Click on the line number.
- Right-click the line number and select "Add Breakpoint".

All set breakpoints are displayed in the debugging area under "Breakpoints".

Linking breakpoints to conditions

To link a breakpoint to a condition, proceed as follows:

1. Open the shortcut menu of the relevant line.
2. Select the entry "Add conditional breakpoint".
   Execution of the script is stopped at the breakpoint when the condition is fulfilled.

Edit conditions as follows:

1. Open the shortcut menu of the relevant line.
2. Select the entry "Edit breakpoint...".

To prevent the script from pausing at a selected line, proceed as follows:

1. Open the shortcut menu of the respective line.
2. Select the entry "Never pause here".

Showing and hiding breakpoints

When you hide a breakpoint, its position is retained. The script then ignores the hidden breakpoint. When you need the breakpoint again, it can simply be shown.

In the debugging area, all breakpoints set in the selected group are displayed under "Breakpoints".

You have several options to show a breakpoint:

- Set the check mark in front of the relevant breakpoint in the debugging area under "Breakpoints".
- Alternatively, right-click the number of the respective line in the code display area and then select "Enable breakpoint".

You have several options to hide a breakpoint:

- Remove the check mark in front of the relevant breakpoint in debugging area under "Breakpoints".
- Alternatively, right-click the number of the respective line in the code display area and then select "Disable breakpoint".

To show or hide all breakpoints, follow these steps:

1. Open the shortcut menu in the debugging area under "Breakpoints".
2. Select "Enable all breakpoints" or "Disable all breakpoints"

Enabling and disabling breakpoints

You can enable or disable all breakpoints independent of showing or hiding individual breakpoints.

You have several options to enable or disable all breakpoints:

- Click on the  "Deactivate breaktpoints" button in the debugging area.
- Open the shortcut menu of a breakpoint in the debugging area and select "Activate breakpoints" or "Deactivate breakpoints".
- Press <Ctrl + F8>.

Deleting breakpoints

You have several options to delete a breakpoint:

- Click on the breakpoint in the code display area.
- Open the shortcut menu of the breakpoint in the code display area and select "Remove breakpoint".
- Open the shortcut menu in the debugging area under "Breakpoints" and select "Remove breakpoint"..

To delete breakpoints, the shortcut menu offers the following additional options in the debugging area under "Breakpoints":

- Delete all breakpoints ("Remove all breakpoints")
- Delete all breakpoints except the selected breakpoint ("Remove other breakpoints")

10.1.7.6 Step-by-step execution of scripts

Introduction

The following options are available to execute your script step-by-step:

- Execute script to the next breakpoint
- Force execution of a script
- Execute script to the next function call
- Jump into a function
- Jump out of a function
- Execute script up to a selected line
- Pause at Exceptions
- Use call stack

Requirement

- The group you want to debug is selected.
- The script pauses at a breakpoint.

Execute script to the next breakpoint

To pause the continuation of a script, you have several options:

- Click on the "Resume script execution" button in the debugging area.
- Press the <F8> key.
  The script is executed to the next breakpoint. If there is no other breakpoint, the script is executed completely.

Force execution of a script

To ignore the following breakpoints when resuming execution of a paused script, proceed as follows:

1. Click and hold down the "Resume script execution" button.
   The "Force script execution" button appears.
2. Move the mouse pointer to the "Force script execution" button while keeping the mouse button pressed.
3. Now release the mouse button.
   The script is executed to the end.

Execute script to the next function call

If a line with a breakpoint contains a function that you are not interested in, you can suppress the debugging of this function:

- Click on the "Step over next function call" button in the debugging area.
- Press the <F10> key.
  The function is executed without the script pausing within the function.

Jumping into a function

If the script pauses in a line containing a function that interests you, you can pause the script in that function:

- Click on the "Step into next function call" button in the debugging area.
- Press the <F11> key.
  The script pauses in the first line of the function.

Note

You can only jump into functions that you have defined yourself.

Jump out of a function

If the script pauses within a function that you are not interested in, you can suppress further debugging of this function:

- Click on the "Step out of current function" button in the debugging area.
- Press the key combination <Shift + F11>.

Note

You can only jump out of a function that you have defined yourself.

Execute script up to a selected line

To pause a paused script again at a selected line, proceeds as follows:

1. Right-click the number of the line in the code display area.
2. Select the entry "Continue to here".
   The script pauses at the selected line.

Pause at Exceptions

- To pause the script at Exceptions, click on the "Pause on exceptions" button in the debugging area.

Use call stack

- To jump into a function of the call stack, click on the corresponding entry under "Call Stack".

Note

You can only jump into functions that you have defined yourself.

10.1.7.7 Show values

Introduction

To identify errors in your script efficiently, have current values displayed while the script is being executed. This way you can view properties of objects or parameters of functions, for example. You can find additional information on objects and their properties under "WinCC Unified Object Model".

Requirements

- The group you want to debug is selected.
- The script pauses at a breakpoint.

Procedure

You view values by moving the mouse over the label in the code display area.

You also have the following options to view values:

- In the debugging area under "Scope"
- In the debugging area under "Watch"
- In the console

"Scope" area

All local values ("Local"), functions ("Module") and global values ("Global") that are defined at this time are displayed in the "Scope" area.

The values cannot be edited.

"Watch" area

In the "Watch" area, you view how values change in the course of a script.

The following buttons are available to you:

- "Add expression": Add a value
- "Refresh": Refresh the "Watch" area
- "Delete watch expression": Delete a value from the "Watch" area. Available when the mouse pointer is located above the respective value.

Console

The values available at the current time can be called in the console.

- You show or hide the console with <Esc>.

Call the current values in the console as follows:

1. Enter the name of a local or global value in the console.
2. Press <Enter>.

See also

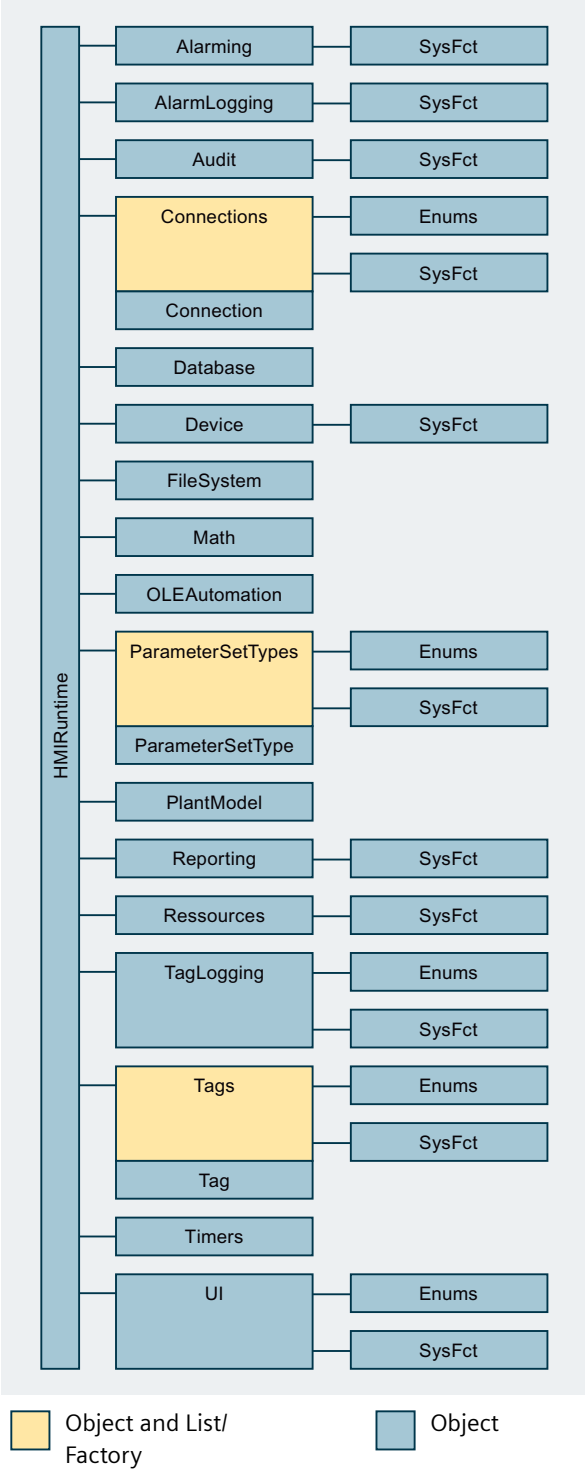
WinCC Unified object model (Page 1037)

10.2 WinCC Unified object model

10.2.1 WinCC Unified object model

Object model

The figure below shows an overview of the object model of the graphical Runtime system of WinCC Unified:



You access the objects of the graphical runtime system through the object model.

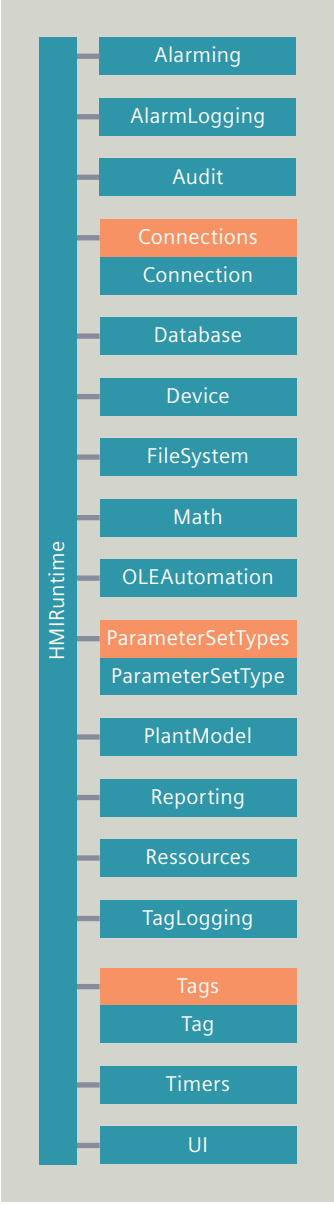
Use

You use the object model as follows:

- Objects
  Objects and lists give you access to basic elements of the runtime system, for example, tags, screens and levels.
- Properties
  You use the properties to read the current status of individual objects, for example, the name of a tag. You can also change many properties of the objects directly, for example, enable a button.
- Methods
  You apply methods to individual objects and write, for example, tag values back to the AS or output alarms in runtime.

10.2.2 HMIRuntime

Description



The "HMIRuntime" object represents the runtime system of WinCC Unified. The "HMIRuntime" object contains properties, methods and all objects of the runtime that support scripting.

Use

Note

The following objects have an alias to allow briefer notation:

- Alias `Tags` corresponds to `HMIRuntime.Tags`
- Alias `UI` corresponds to `HMIRuntime.UI`
- Alias `Screen` corresponds to the screen in which the script is executed.
- Alias `PlantModel` corresponds to `HMIRuntime.PlantModel`
- Alias `ParameterSetTypes` corresponds to `HMIRuntime.ParameterSetTypes`
- Alias `Faceplate` corresponds to the faceplate type in which the script is executed.

Object type

HMIRuntime

Properties

The "HMIRuntime" object has the following properties:

Language
Sets the runtime language.

Methods

The "HMIRuntime" object has the following methods:

- **GetDetailedErrorDescription()**
  Returns a detailed error description of the error code.
- **Trace()**
  Outputs a user-defined text via the debug output of the trace viewer in runtime.

10.2.2.1 HMIRuntime.Language

Description

The "Language" property specifies the runtime language.

Type

UInt32, HMILCID

Access

Read-write

Syntax

```
HMIRuntime.Language
```

Locale IDs (HMILCID)

The following table contains the Microsoft locale IDs of the languages supported in runtime:

Language	Country/Region	Locale ID
Afrikaans	South Africa	1078
Albanian	Albania	1052
Armenian	Armenia	1067
Azerbaijani (Cyrillic)	Azerbaijan	2092
Azerbaijani (Latin)	Azerbaijan	1068
Basque	Basque country	1069
Belarusian	Belarus	1059
Bulgarian	Bulgaria	1026
Chinese	Hong Kong S.A.R.	3076
Chinese	Macao S.A.R.	5124
Chinese	Singapore	4100
Chinese	Taiwan	1028
Chinese	PR China	2052
Danish	Denmark	1030
German	Germany	1031
German	Liechtenstein	5127
German	Luxembourg	4103
German	Austria	3079
German	Switzerland	2055
English	Australia	3081
English	Belize	10249
English	United Kingdom	2057
English	Ireland	6153
English	Jamaica	8201
English	Canada	4105
English	Caribbean	9225
English	New Zealand	5129
English	Philippines	13321
English	Zimbabwe	12297
English	South Africa	7177
English	Trinidad and Tobago	11273
English	USA	1033
Estonian	Estonia	1061
Faroese	Faroe Islands	1080
Finnish	Finland	1035
French	Belgium	2060

Language	Country/Region	Locale ID
French	France	1036
French	Canada	3084
French	Luxembourg	5132
French	Monaco	6156
French	Switzerland	4108
Galician	Galicia	1110
Georgian	Georgia	1079
Greek	Greece	1032
Hindi	India	1081
Indonesian	Indonesia	1057
Icelandic	Iceland	1039
Italian	Italy	1040
Italian	Switzerland	2064
Japanese	Japan	1041
Kazakh	Kazakhstan	1087
Catalan	Catalonia	1027
Kyrgyz	Kyrgyzstan	1088
Konkani	India	1111
Korean	Korea	1042
Croatian	Croatia	1050
Latvian	Latvia	1062
Malay	Brunei Darussalam	2110
Malay	Malaysia	1086
Macedonian	Macedonia, FYRM	1071
Mongolian (Cyrillic)	Mongolia	1104
Dutch	Belgium	2067
Dutch	Netherlands	1043
Norwegian (Bokmål)	Norway	1044
Norwegian (Nynorsk)	Norway	2068
Polish	Poland	1045
Portuguese	Brazil	1046
Portuguese	Portugal	2070
Romanian	Romania	1048
Russian	Russia	1049
Sanskrit	India	1103
Swedish	Finland	2077
Swedish	Sweden	1053
Serbian (Cyrillic)	Serbia and Montenegro (formerly)	3098
Serbian (Latin)	Serbia and Montenegro (formerly)	2074
Slovakian	Slovakia	1051
Slovenian	Slovenia	1060

Language	Country/Region	Locale ID
Spanish	Argentina	11274
Spanish	Bolivia	16394
Spanish	Chile	13322
Spanish	Costa Rica	5130
Spanish	Dominican Republic	7178
Spanish	Ecuador	12298
Spanish	El Salvador	17418
Spanish	Guatemala	4106
Spanish	Honduras	18442
Spanish	Colombia	9226
Spanish	Mexico	2058
Spanish	Nicaragua	19466
Spanish	Panama	6154
Spanish	Paraguay	15370
Spanish	Peru	10250
Spanish	Puerto Rico	20490
Spanish	Spain	1034
Spanish	Uruguay	14346
Spanish	Venezuela	8202
Swahili	Kenya	1089
Tatar	Russia	1092
Thai	Thailand	1054
Czech	Czech Republic	1029
Turkish	Turkey	1055
Ukrainian	Ukraine	1058
Hungarian	Hungary	1038
Uzbek (Cyrillic)	Uzbekistan	2115
Uzbek (Latin)	Uzbekistan	1091
Vietnamese	Vietnam	1066

See also

HMIRuntime (Page 1040)

10.2.2.2 HMIRuntime.GetDetailedErrorDescription()

Description

The "GetDetailedErrorDescription" method returns a detailed error description of the error code passed.

Syntax

```
HMIRuntime.GetDetailedErrorDescription(errorCode);
```

Parameters

errorCode

Type: ErrorCode

Error code for which the detailed error description is to be output.

Return value

String

See also

HMIRuntime (Page 1040)

10.2.2.3 HMIRuntime.Trace()

Description

The "Trace" method outputs a user-defined text through the debug output of the trace viewer in runtime.

Syntax

```
HMIRuntime.Trace(message);
```

Parameter

message

Type: String

The text which is output.

Return value

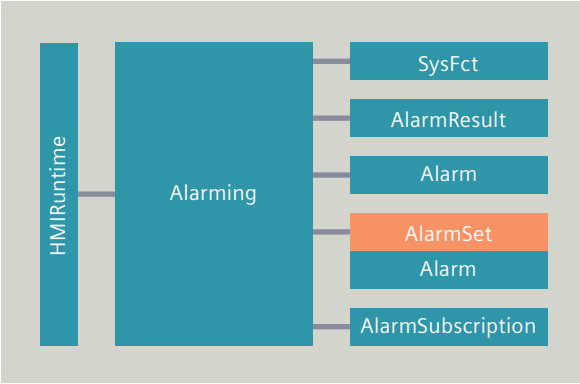
--

See also

HMIRuntime (Page 1040)

10.2.2.4 Alarming

Description



The "Alarming" object ("HMIAlarming" type) enables access to the WinCC Unified alarm system. You can create a new "AlarmSet" list, compile active alarms ("AlarmSubscription" objects) and reference configured alarms ("Alarm" objects) for read and write access.

Object type

HMIAlarming

Properties

--

Methods

The "Alarming" object has the following methods:

- **Alarms()**
  Returns an "Alarm" object.
- **CreateAlarmSet()**
  Creates an "AlarmSet" object.
- **CreateSubscription()**
  Creates an "AlarmSubscription" object.
- **GetActiveAlarms()**
  Supplies all active alarms at the time of the call.

Alarming.Alarms()

Description

The "Alarms" method returns an "Alarm" object ("HMIAlarm" type). You can use the "Alarm" object to perform operations with an alarm, for example, acknowledge or comment.

Syntax

```
HMIRuntime.Alarming.Alarms(AlarmName);
```

Parameters

AlarmName

Type: String, HMIAlarm

Name of an alarm

Return value

Object, HMIAlarm (Page 1050)

Example

Create an "Alarm" object with the "alarm1" alarm:

Copy code

```
var alarm = HMIRuntime.Alarming.Alarms('alarm1');
```

See also

Alarm (Page 1050)

Alarming (Page 1046)

Alarming.CreateAlarmSet()

Description

The "CreateAlarmSet" method creates a new "AlarmSet" object ("HMIAlarmSet" type). With the returned "AlarmSet" object, you can execute operations with multiple alarms in one call, for example, acknowledge or comment.

Syntax

```
HMIRuntime.Alarming.CreateAlarmSet([AlarmNames]);
```

Parameters

`AlarmNames`

Optional, type: String, HMIAlarm | String[], HMIAlarm | Variant[][], HMIAlarm

Name of one or more active alarms added to the "AlarmSet" object. Without parameters, an empty "AlarmSet" object is created.

Return value

Object, HMIAlarmSet (Page 1056)

Example

Create an "AlarmSet" object with two alarms:

Copy code

```
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet(['alarm1', 'alarm2']);
```

See also

Alarming (Page 1046)

AlarmSet (Page 1056)

Alarming.CreateSubscription()

Description

The "CreateSubscription" method creates an "AlarmSubscription" object ("HMIAlarmSubscription" type). With the returned object "AlarmSubscription" you specify the grouping of active alarms.

Syntax

```
HMIRuntime.Alarming.CreateSubscription();
```

Parameters

--

Return value

Object, HMIAlarmSubscription (Page 1067)

See also

Alarming (Page 1046)

AlarmSubscription (Page 1067)

Alarming.GetActiveAlarms()

Description

The "GetActiveAlarms" method returns all active alarms at the time of the call. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call.

Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "AlarmResult" objects or the error code as parameter.

Syntax

```
HMIRuntime.Alarming.GetActiveAlarms(Language[,Filter][,SystemNames])
.then(function(HMIAlarmResult[]) {
    ...
});
.catch(function(ErrorCode) {
    ...
});
```

Parameters

`Language`

Type: UInt32, HMILCID

Language for all texts of an alarm and the filter

`Filter`

Optional, type: String, HMIAlarmFilterString

SQL-type string for filtering

`SystemNames`

Optional, type: String[], HMISystem

A string array containing the SystemNames of the systems by which alarms are to be filtered.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled
  Object, HMIAlarmResult[] (Page 1077) as parameter of the "then()" handler.
- Promise rejected
  ErrorCode as parameter of the "catch()" handler.

Example

Read out all active alarms of the "RUNTIME_1" system:

Copy code

```
var promise = HMIRuntime.Alarming.GetActiveAlarms(1033,"","RUNTIME_1");
```

See also

Alarming (Page 1046)

AlarmResult (Page 1077)

Alarm

Description

The "Alarm" object ("HMIAlarm" type) enables access to properties and methods of active alarms. An "Alarm" object is returned by the "Alarming" or "AlarmSet" lists.

Object type

HMIAlarm

Properties

The "Alarm" object has the following properties:

- **Name**
  Specifies the name of the alarm.
- **ErrorCode**
  Specifies the error code of the last method call of the object.

Methods

The "Alarm" object has the following methods:

- **Acknowledge()**
  Acknowledges active alarms.
- **Disable()**
  Temporarily disables the generation of alarms.
- **Enable()**
  Re-enables disabled alarms for display.
- **Reset()**
  Acknowledges the outgoing state of an active alarm.
- **Shelve()**
  Hides active alarms.
- **Unshelve()**
  Makes hidden active alarms visible again.

Alarm.ErrorCode

Description

The "ErrorCode" property specifies the error code of the last method call of the "Alarm" object.

Type

ErrorCode

Access

Read-write

Syntax

```
Alarm.ErrorCode
```

See also

Alarm (Page 1050)

Alarm.Name

Description

The "Name" property specifies the name of the alarm ("Alarm" object).

Type

String, HMIAlarm

Access

Read-write

Syntax

```
Alarm.Name
```

See also

Alarm (Page 1050)

Alarm.Acknowledge()

Description

The "Acknowledge" method acknowledges active alarms. The method uses the entire "Alarm" object as reference.

The method is executed as a synchronous operation.

Syntax

```
Alarm.Acknowledge([InstanceID])
```

Parameters

`InstanceID`

Optional, type: UInt32

Number of the alarm instance.

Return value

ErrorCode

Example

Acknowledge an alarm:

Copy code

```
var AckErrorcode = alarm.Acknowledge();
```

See also

Alarm (Page 1050)

Alarm.Disable()

Description

The "Disable" method temporarily deactivates the generation of alarms in the alarm source. You can reactivate the generation of the alarms with the "Enable" method.

You can use the "Disable" method to prevent the display of the alarms, for example, for maintenance work.

The method is executed as a synchronous operation.

Syntax

```
Alarm.Disable()
```

Parameters

--

Return value

ErrorCode

Example

Deactivate an alarm:

Copy code

```
var DisableErrorcode = alarm.Disable();
```

See also

Alarm (Page 1050)

Alarm.Enable() (Page 1053)

Alarm.Enable()

Description

The "Enable" method re-enables disabled alarms for display. You can temporarily deactivate the generation of alarms in the alarm source with the "Disable" method.

The method is executed as a synchronous operation.

Syntax

```
Alarm.Enable()
```

Parameters

--

Return value

ErrorCode

Example

Log an alarm:

Copy code

```
var EnableErrorcode = alarm.Enable();
```

See also

Alarm (Page 1050)

Alarm.Disable() (Page 1053)

Alarm.Reset()

Description

The "Reset" method acknowledges the outgoing state of an active alarm. The alarm is removed from the alarm system. The method uses the entire "Alarm" object as reference.

The method is executed as a synchronous operation.

Syntax

```
Alarm.Reset([InstanceID])
```

Parameters

`InstanceID`

Optional, type: UInt32

Number of the alarm instance.

Return value

ErrorCode

See also

Alarm (Page 1050)

Alarm.Shelve()

Description

The "Shelve" method hides active alarms. These are no longer displayed by the alarm control in runtime. You can show the alarms again with the "Unshelve" method.

The method is executed as a synchronous operation.

Syntax

```
Alarm.Shelve()
```

Parameters

--

Return value

ErrorCode

Example

Hides an alarm:

Copy code

```
var Errorcode = alarm.Shelve();
```

See also

Alarm (Page 1050)

Alarm.Unshelve() (Page 1055)

Alarm.Unshelve()

Description

The "Unshelve" method makes active alarms visible again. These are displayed again by the alarm control in runtime. You can hide the alarms again with the "Shelve" method.

The method is executed as a synchronous operation.

Syntax

```
Alarm.Unshelve()
```

Parameters

--

Return value

ErrorCode

See also

Alarm (Page 1050)

Alarm.Shelve() (Page 1055)

AlarmSet

Description

The "AlarmSet" object ("HMIAlarmSet" type) is a list of "Alarm" objects that provides optimized access to active alarms in runtime. After the initialization of the "AlarmSet" object, you can execute operations with multiple alarms in one call, such as acknowledge or comment. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

You create a new "AlarmSet" object with the "Alarming.CreateAlarmSet" method.

Object type

HMIAlarmSet

Properties

The "AlarmSet" object has the following properties:

- **Count**
  Returns the number of elements of the "AlarmSet" list.

Methods

The "AlarmSet" object has the following methods:

- **Acknowledge()**
  Acknowledges active alarms.
- **Add()**
  Adds alarms to the "AlarmSet" list.

- **Disable()**
  Temporarily disables the generation of alarms.
- **Enable()**
  Re-enables disabled alarms for display.
- **Item()**
  Returns an alarm of the "AlarmSet" list.
- **Remove()**
  Removes alarms by their name from the "AlarmSet" list.
- **Reset()**
  Acknowledges the outgoing state of active alarms.
- **Shelve()**
  Hides active alarms.
- **Unshelve()**
  Makes hidden active alarms visible again.

See also

Alarming.CreateAlarmSet() (Page 1047)

AlarmSet.Count

Description

The "Count" property returns the number of elements in the "AlarmSet" list.

Type

UInt32

Access

Read-only

Syntax

```
AlarmSet.Count
```

See also

AlarmSet (Page 1056)

AlarmSet.Acknowledge()

Description

The "Acknowledge" method acknowledges active alarms. The method uses the entire "Alarm" object as reference.

The method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called with the ErrorCode as parameter after execution.

Syntax

```
AlarmSet.Acknowledge()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

--

Return value

Promise

ErrorCode, depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise rejected
  ErrorCode as parameter of the "catch()" handler

Example

Acknowledge all alarms of an "AlarmSet" object:

Copy code

```
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet('alarm1', 'alarm2', 'alarm3');
var promise = alarmSet.Acknowledge();
```

See also

AlarmSet (Page 1056)

AlarmSet.Add()

Description

The "Add" method adds alarms to the "AlarmSet" list. The alarms are referenced by name.

Syntax

```
AlarmSet.Add(AlarmName,[InstanceID])
```

Parameters

`AlarmName`

Type: String, HMIAlarm | String[], HMIAlarm | Variant[][], HMIAlarm

Names of "Alarm" objects that are added to the list.

The following data types are supported:

- Alarm name
- Array with alarm names
- Two-dimensional array with alarm name/value pairs

`InstanceID`

Optional, type: UInt32

Number of the alarm instance.

Return value

Object[], HMIAlarm (Page 1050)

Example

Create an "AlarmSet" object with two alarms:

Copy code

```
var name1 =  ResultSet[index].Name;
var name2 =  ResultSet[index+1].Name;
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet();
var addedAlarms = alarmSet.Add([name1,name2]);
```

See also

AlarmSet (Page 1056)

Alarm (Page 1050)

AlarmSet.Disable()

Description

The "Disable" method temporarily deactivates the generation of alarms in the alarm source. You can reactivate the generation of the alarms with the "Enable" method.

You can use the "Disable" method to prevent the display of the alarms, for example, for maintenance work.

The method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Syntax

```
AlarmSet.Disable()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

--

Return value

Promise

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise rejected
  ErrorCode as parameter of the "catch()" handler

Example

Deactivate all alarms of an "AlarmSet" object:

Copy code

```
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet('alarm1', 'alarm2');
var promise = alarmSet.Disable();
```

See also

AlarmSet (Page 1056)

AlarmSet.Enable() (Page 1061)

AlarmSet.Enable()

Description

The "Enable" method re-enables disabled alarms for display. You can temporarily deactivate the generation of alarms in the alarm source with the "Disable" method.

The method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Syntax

```
AlarmSet.Enable()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

--

Return value

Promise

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise rejected
  ErrorCode as parameter of the "catch()" handler

Example

Activate all alarms of an "AlarmSet" object:

Copy code

```
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet('alarm1', 'alarm2');
var promise = alarmSet.Enable();
```

See also

AlarmSet (Page 1056)

AlarmSet.Disable() (Page 1060)

AlarmSet.Item()

Description

The "Item" method returns an "Alarm" object of the "AlarmSet" list.

Syntax

```
AlarmSet[.Item](name,[InstanceID])
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "AlarmSet" object.

Parameters

`name`

Type: String, HMIAlarm | Int32, HMIAlarm

Name or index number (1...n) of an "Alarm" object in the list

Note

The index number of an "Alarm" object does not represent the order in which the "Alarm" objects were added to the "AlarmSet" list.

`InstanceID`

Optional, type: UInt32

Number of the alarm instance

Return value

Object, HMIAlarm (Page 1050)

See also

AlarmSet (Page 1056)

Alarm (Page 1050)

AlarmSet.Remove()

Description

The "Remove" method removes alarms ("Alarm" objects) from the "AlarmSet" list by their names.

Syntax

```
AlarmSet.Remove(AlarmName,[InstanceID])
```

Parameters

AlarmName

Type: String, HMITag | String[], HMITag

Name of "Alarm" objects that are removed from the "AlarmSet" list.

The following data types are supported:

- Alarm name
- Array with alarm names

Note

No "Alarm" object can be transferred as a parameter. An "Alarm" object is referenced using the name.

InstanceID

Optional, type: UInt32

Number of the alarm instance

Return value

ErrorCode

Example

Create an "AlarmSet" object with two alarms and then remove both alarms from the AlarmSet again:

Copy code

```
var name1 =  ResultSet[index].Name;
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet([name1, 'alarm2']);
alarmSet.remove([name1,'alarm2']);
```

See also

AlarmSet (Page 1056)

AlarmSet.Reset()

Description

The "Reset" method acknowledges the outgoing state of active alarms. The alarms are removed from the alarm system. The method uses the entire "Alarm" object as reference.

The method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Syntax

```
AlarmSet.Reset()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

--

Return value

Promise

ErrorCode, depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

AlarmSet (Page 1056)

AlarmSet.Shelve()

Description

The "Shelve" method hides active alarms. These are no longer displayed by Alarm Control in runtime. You can show the alarms again with the "Unshelve" method.

The method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Syntax

```
AlarmSet.Shelve()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

--

Return value

Promise

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise rejected
  ErrorCode as parameter of the "catch()" handler

Example

Hide all alarms of an "AlarmSet" object:

Copy code

```
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet('alarm1', 'alarm2');
var promise = alarmSet.Shelve();
```

See also

AlarmSet (Page 1056)

AlarmSet.Unshelve() (Page 1066)

AlarmSet.Unshelve()

Description

The "Unshelve" method makes active alarms visible again. These are once again displayed by the Alarm Control in runtime. You can hide the alarms again with the "Shelve" method.

The method is applied to all "Alarm" objects of the list. The method is executed as asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the operation. Depending on the result, the corresponding handler of the Promise pattern with the ErrorCode as parameter is called after the execution.

Syntax

```
AlarmSet.Unshelve()
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

--

Return value

Promise

ErrorCode, for "AlarmSet" objects depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise rejected
  ErrorCode as parameter of the "catch()" handler

Example

Show all alarms of an "AlarmSet" object again:

Copy code

```
var alarmSet = HMIRuntime.Alarming.CreateAlarmSet('alarm1', 'alarm2');
var promise = alarmSet.Unshelve();
```

See also

AlarmSet (Page 1056)

AlarmSet.Shelve() (Page 1065)

Alarm

Description

Alarm (Page 1050)

AlarmSubscription

Description

The "AlarmSubscription" object ("HMIAlarmSubscription" type) enables access to active alarms.

Use

The "AlarmSubscription" object represents a selection of active alarms. An "AlarmSubscription" object is initialized through the "CreateSubscription" method of the "Alarming" object. The active alarms are then grouped and called according to the properties of the "AlarmSubscription" object. Finally, notification is given of the changes to the alarm image.

Object type

HMIAlarmSubscription

Properties

The "AlarmSubscription" object has the following properties:

- **Filter**
  Defines a string for filtering active alarms.
- **Language**
  Specifies the current runtime language.
- **OnAlarm**
  Sets the name of the callback function "OnAlarm" for monitoring active alarms.
- **SystemNames**
  Specifies the name of the runtime for compiling active alarms.

Methods

The "AlarmSubscription" object has the following methods:

- **Start()**
  Activates the monitoring of defined alarms of the "AlarmSubscription" object.
- **Stop()**
  Stops monitoring of defined alarms of the "AlarmSubscription" object.

See also

Alarming.CreateSubscription() (Page 1048)

AlarmSubscription.Filter

Description

The "Filter" property specifies a string for filtering active alarms.

The syntax of the filter string corresponds to the WHERE clause of an SQL command.

Type

String, HMIAlarmFilterString

Access

Read-write

Syntax

```
AlarmSubscription.Filter
```

Supported alarm properties

The following properties of an alarm can be used in the filter string:

- AcknowledgementTime
- Alarm
- AlarmClassName
- AlarmClassSymbol
- AlarmParameterValues
- AlarmText1 ... 9
- Area
- BackColor
- ChangeReason
- ClearTime
- Connection
- EventText
- Flashing
- InfoText
- InstanceID
- LoopInAlarm

- ModificationTime
- Name
- Origin
- Priority
- RaiseTime
- ResetTime
- SourceID
- SourceType
- State
- StateMachine
- StateText
- SuppressionState
- SystemSeverity
- TextColor
- Value
- ValueLimit

Operators

The following operators can be used in the filter string:

Operator	Description	Example
`=`	equal to	`AlarmClassName = 'demo'`
`IS NOT string`	is not equal to the string *string*	`AlarmText4 IS NOT 'Text5'`
`<>`	not equal	`Value <> 0.0`
`>`	greater than	`ModificationTime > '11.08.2016'`
`<`	less than	`Value < 75.0`
`>=`	greater than or equal to	`Value >= 25.0`
`<=`	less than or equal to	`Value <= 75.0`
`OR, ||`	logical OR	`State = 1 OR State = 3`
`AND, &&`	logical AND	`EventText = 'Text1' AND Origin = 'Motor'`
`BETWEEN`	within a range	`Value BETWEEN 25.0 AND 75.0`
`NOT BETWEEN`	outside a range	`Value NOT BETWEEN 25.0 AND 75.0`
`LIKE string`	corresponds to the string *string*	`Name LIKE 'Motor*'`
`NOT LIKE string`	does not correspond to the string *string*	`Name NOT LIKE 'Valve*'`
`IN (v1, v2, …)`	corresponds to one or more values	`State IN (1, 4, 7)`
`NOT IN (v1, v2, …)`	does not correspond to one or more values	`State NOT IN (0, 2, 3, 5, 6)`

Operator	Description	Example
`IS NULL`	compares to zero (missing data)	`Context IS NULL`
`IS NOT NULL`	compares to zero (unknown data)	`Context IS NOT NULL`

Wildcards

The following wildcards can be used for characters of filter strings:

Wildcard	Description	Example
`*`	replaces 0, 1 or more characters	`Name LIKE 'Motor*'`
`?`	replaces exactly 1 character	`Name = 'Recipe?'`

See also

AlarmSubscription (Page 1067)

AlarmSubscription.Language

Description

The "Language" property specifies the current language for all texts of an alarm and the filter.

Type

UInt32, HMILCID

Access

Read-write

Syntax

```
AlarmSubscription.Language
```

Locale IDs (HMILCID)

The following table contains the Microsoft locale IDs of the languages supported in runtime:

Language	Country/Region	Locale ID
Afrikaans	South Africa	1078
Albanian	Albania	1052
Armenian	Armenia	1067
Azerbaijani (Cyrillic)	Azerbaijan	2092
Azerbaijani (Latin)	Azerbaijan	1068
Basque	Basque country	1069
Belarusian	Belarus	1059

Language	Country/Region	Locale ID
Bulgarian	Bulgaria	1026
Chinese	Hong Kong S.A.R.	3076
Chinese	Macao S.A.R.	5124
Chinese	Singapore	4100
Chinese	Taiwan	1028
Chinese	PR China	2052
Danish	Denmark	1030
German	Germany	1031
German	Liechtenstein	5127
German	Luxembourg	4103
German	Austria	3079
German	Switzerland	2055
English	Australia	3081
English	Belize	10249
English	United Kingdom	2057
English	Ireland	6153
English	Jamaica	8201
English	Canada	4105
English	Caribbean	9225
English	New Zealand	5129
English	Philippines	13321
English	Zimbabwe	12297
English	South Africa	7177
English	Trinidad and Tobago	11273
English	USA	1033
Estonian	Estonia	1061
Faroese	Faroe Islands	1080
Finnish	Finland	1035
French	Belgium	2060
French	France	1036
French	Canada	3084
French	Luxembourg	5132
French	Monaco	6156
French	Switzerland	4108
Galician	Galicia	1110
Georgian	Georgia	1079
Greek	Greece	1032
Hindi	India	1081
Indonesian	Indonesia	1057
Icelandic	Iceland	1039
Italian	Italy	1040
Italian	Switzerland	2064
Japanese	Japan	1041

Language	Country/Region	Locale ID
Kazakh	Kazakhstan	1087
Catalan	Catalonia	1027
Kyrgyz	Kyrgyzstan	1088
Konkani	India	1111
Korean	Korea	1042
Croatian	Croatia	1050
Latvian	Latvia	1062
Malay	Brunei Darussalam	2110
Malay	Malaysia	1086
Macedonian	Macedonia, FYRM	1071
Mongolian (Cyrillic)	Mongolia	1104
Dutch	Belgium	2067
Dutch	Netherlands	1043
Norwegian (Bokmål)	Norway	1044
Norwegian (Nynorsk)	Norway	2068
Polish	Poland	1045
Portuguese	Brazil	1046
Portuguese	Portugal	2070
Romanian	Romania	1048
Russian	Russia	1049
Sanskrit	India	1103
Swedish	Finland	2077
Swedish	Sweden	1053
Serbian (Cyrillic)	Serbia and Montenegro (formerly)	3098
Serbian (Latin)	Serbia and Montenegro (formerly)	2074
Slovakian	Slovakia	1051
Slovenian	Slovenia	1060
Spanish	Argentina	11274
Spanish	Bolivia	16394
Spanish	Chile	13322
Spanish	Costa Rica	5130
Spanish	Dominican Republic	7178
Spanish	Ecuador	12298
Spanish	El Salvador	17418
Spanish	Guatemala	4106
Spanish	Honduras	18442
Spanish	Colombia	9226
Spanish	Mexico	2058
Spanish	Nicaragua	19466
Spanish	Panama	6154
Spanish	Paraguay	15370

Language	Country/Region	Locale ID
Spanish	Peru	10250
Spanish	Puerto Rico	20490
Spanish	Spain	1034
Spanish	Uruguay	14346
Spanish	Venezuela	8202
Swahili	Kenya	1089
Tatar	Russia	1092
Thai	Thailand	1054
Czech	Czech Republic	1029
Turkish	Turkey	1055
Ukrainian	Ukraine	1058
Hungarian	Hungary	1038
Uzbek (Cyrillic)	Uzbekistan	2115
Uzbek (Latin)	Uzbekistan	1091
Vietnamese	Vietnam	1066

See also

AlarmSubscription (Page 1067)

AlarmSubscription.OnAlarm

Description

The "OnAlarm" property specifies the name of the "OnAlarm" callback function for monitoring active alarms.

The properties of the active alarms are passed to the "OnAlarm" callback function as the "AlarmResultArray" object.

Required prototype of the callback function:
`OnAlarm(errorCode,systemName,alarmResultArray)`

Type

Function, HMIOnAlarmCB

Access

Write-only

Syntax

```
AlarmSubscription.OnAlarm
```

Parameters

Parameters of the callback function:

errorCode

Type: ErrorCode

Error code of the active alarm

systemName

Type: String

Name of the runtime system

alarmResultArray

Type: Object, HMIAlarmResult[]

Array with "AlarmResult" objects of the active alarm

Example

Monitor active alarms and output their properties:

Copy code

```
function AlarmSubscribe() {
    var subs = HMIRuntime.Alarming.CreateSubscription();
    subs.Filter = "";
    subs.language = 1033;
    subs.SystemNames = "System4";
    subs.OnAlarm = OnAlarm;
    subs.Start();
}

function OnAlarm(Errorcode, SystemName, ResultArray) {
    HMIRuntime.Trace("Script OnAlarm Called");
    var alarmcount = ResultArray.length;

    for (var index = 0; index < alarmcount; ++index) {
        if(ErrorCode[index] >= 0) //access resultarray only when it's
succeeded
        {
            HMIRuntime.Trace(ResultArray[index].EventText);
            HMIRuntime.Trace(ResultArray[index].InstanceID);
            HMIRuntime.Trace(ResultArray[index].AlarmClassName);
            HMIRuntime.Trace(ResultArray[index].State);
            HMIRuntime.Trace(SystemName[index]);
            HMIRuntime.Trace(Errorcode[index]);
        }
    }
}
```

See also

AlarmSubscription (Page 1067)

AlarmSubscription.Start() (Page 1075)

AlarmSubscription.SystemNames

Description

The "SystemNames" property specifies the name of the runtime ("HMISystem" type) for compiling active alarms.

Type

String[], HMISystem

Access

Read-write

Syntax

```
AlarmSubscription.SystemNames
```

See also

AlarmSubscription (Page 1067)

AlarmSubscription.Start()

Description

The "Start" method activates the monitoring of defined alarms of the "AlarmSubscription" object.

Syntax

```
AlarmSubscription.Start()
```

Parameters

--

Return value

ErrorCode

Example

Start monitoring of alarms and output number of alarms:

Copy code

```
var subs =  HMIRuntime.Alarming.CreateSubscription();
subs.SystemNames = "System24";
subs.Language = 1033;
subs.OnAlarm = function (Errorcode, sysID, ResultSet){
    HMIRuntime.Trace("Script OnAlarm Called");
    var alarmCount = ResultSet.length;
    HMIRuntime.Trace("Alarm count = " + alarmCount);
};
subs.Start();
```

See also

AlarmSubscription (Page 1067)

AlarmSubscription.OnAlarm (Page 1073)

AlarmSubscription.Stop() (Page 1076)

AlarmSubscription.Stop()

Description

The "Stop" method stops the monitoring of defined alarms of the "AlarmSubscription" object.

Syntax

```
AlarmSubscription.Stop()
```

Parameters

--

Return value

ErrorCode

Example

Cancel monitoring of alarms after the first output:

Copy code

```
var subs =  HMIRuntime.Alarming.CreateSubscription();
subs.SystemNames = "System24";
subs.Language = 1033;
subs.OnAlarm = function (Errorcode, sysID, ResultSet){
    HMIRuntime.Trace("Script OnAlarm Called");
    var alarmCount = ResultSet.length;
    HMIRuntime.Trace("Alarm count = " + alarmCount);
    subs.Stop();
};
subs.Start();
```

See also

AlarmSubscription (Page 1067)

AlarmSubscription.Start() (Page 1075)

AlarmResult

Description

The "AlarmResult" object ("HMIAlarmResult" type) provides access to the properties of an active alarm. The "AlarmResult" object is a pure data object which maps all properties of an active alarm.

Use

The OnAlarm callback function is called after an active alarm has been output. The "AlarmResult" object together with the SystemName and ErrorCode are transferred as parameters to this function.

In the case of multiple active alarms, the parameters are passed as lists to the OnAlarm callback function. Alarms from different servers are processed in only one call of the OnAlarm callback function.

The "AlarmResult" object only contains properties and no methods.

All texts of the "AlarmResult" object are monolingual strings. The language is specified with the "AlarmSubscription.Language" property.

Object type

HMIAlarmResult

Properties

The "AlarmResult" object has the following properties:

- **AcknowledgementTime**
  Returns the time when an alarm was acknowledged.
- **AlarmClassName**
  Returns the name of the alarm class.
- **AlarmClassSymbol**
  Returns the abbreviation for the display of the alarm class.
- **AlarmGroupID**
  Returns the alarm group ID.
- **AlarmParameterValues**
  Returns an array with parameter values of an alarm.
- **AlarmText**
  Returns the localized additional texts 1-9 of an alarm as array.
- **Area**
  Returns the area of origin of an alarm.
- **BackColor**
  Returns the background color of an alarm.
- **ChangeReason**
  Returns the event that triggered the change in the alarm state.
- **ClearTime**
  Returns the time of the reset of an alarm.
- **Connection**
  Returns the name of the connection via which an alarm was triggered.
- **Duration**
  Returns the duration of an alarm.
- **EventText**
  Returns a localized text describing an event to the alarm.
- **Flashing**
  Returns whether the specified object flashes in runtime.
- **ID**
  Returns the ID of the alarm that is also used in the display.
- **InstanceID**
  Returns the ID of alarms with multiple instances.
- **InvalidFlags**
  Returns the cause of invalid data of an alarm.
- **LoopInAlarm**
  Returns the name of the function that navigates from the display of an active alarm to its origin.
- **LoopInAlarmParameterValues**
  Returns the parameters of the function that navigates from the display of an active alarm to its origin.

- **ModificationTime**
  Returns the time stamp of the last change of the alarm state.
- **Name**
  Returns the name of the alarm.
- **NotificationReason**
  Returns the reason of an alarm.
- **Origin**
  Returns the origin of an alarm.
- **Priority**
  Returns the relevance of an alarm or a machine status.
- **RaiseTime**
  Returns the raise time of an alarm.
- **ResetTime**
  Returns the time of the reset of an alarm.
- **SourceID**
  Returns the source where an alarm was triggered.
- **State**
  Returns the state of an alarm.
- **StateText**
  Returns the alarm state as text, for example, "Incoming" or "Outgoing".
- **SuppressionState**
  Returns the visibility status of an active alarm.
- **SystemSeverity**
  Returns the level of the severity of a system fault as property of a system alarm.
- **TextColor**
  Returns the text color of the alarm state.
- **UserName**
  Returns the name of the user who triggered the alarm.
- **UserResponse**
  Returns the expected or required user response to an alarm.
- **Value**
  Returns the process value of an alarm.
- **ValueLimit**
  Returns the limit of a process value of an alarm.
- **ValueQuality**
  Returns the quality level of a process value of an alarm.

Methods

--

See also

Alarming.GetActiveAlarms() (Page 1049)

AlarmResult.AcknowledgementTime

Description

The "AcknowledgementTime" property returns the time of the acknowledgment of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
AlarmResult.AcknowledgementTime
```

See also

AlarmResult (Page 1077)

AlarmResult.AlarmClassName

Description

The "AlarmClassName" property returns the name of the alarm class.

Type

String

Access

Read-only

Syntax

```
AlarmResult.AlarmClassName
```

See also

AlarmResult (Page 1077)

AlarmResult.AlarmClassSymbol

Description

The "AlarmClassSymbol" property returns the abbreviation for the display of the alarm class of the alarm, for example, "W" for the alarm class "Warning".

Type

String

Access

Read-only

Syntax

```
AlarmResult.AlarmClassSymbol
```

See also

AlarmResult (Page 1077)

AlarmResult.AlarmGroupID

Description

The "AlarmGroupID" property returns the alarm group ID.

Type

UInt8

Access

Read-only

Syntax

```
AlarmResult.AlarmGroupID
```

See also

AlarmResult (Page 1077)

AlarmResult.AlarmParameterValues

Description

The "AlarmParameterValues" property returns an array with parameter values of an alarm.

The parameter values are added to an alarm from the source in the "Incoming" and "Reset" state. They can also include diagnostic or raw data from the PLC in addition to simple tag values from the configured AlarmParamterTags.

Type

Variant[]

Access

Read-only

Syntax

```
AlarmResult.AlarmParameterValues
```

See also

AlarmResult (Page 1077)

AlarmResult.AlarmText

Description

The "AlarmText" property returns the localized additional texts 1-9 as an array. The text can contain triggered placeholders and reference all "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".

Type

String[]

Access

Read-only

Syntax

```
AlarmResult.AlarmText
```

See also

AlarmResult (Page 1077)

AlarmResult.Area

Description

The "Area" property specifies the area of origin of an alarm.

The "Area" property can be configured and, together with the "Origin" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

The "Area" property, for example, includes subsystem, application name or PLC ID. You can sort and filter alarms through the "Area" context.

Type

String

Access

Read-only

Syntax

```
AlarmResult.Area
```

See also

AlarmResult (Page 1077)

AlarmResult.BackColor

Description

The "BackColor" property returns the background color of an alarm.

Type

UInt32

Access

Read-only

Syntax

```
AlarmResult.BackColor
```

See also

AlarmResult (Page 1077)

AlarmResult.ChangeReason

Description

The "ChangeReason" property returns the event that triggered the change in the alarm state. The time of last modification is saved in the "ModificationTime" property.

The alarm status may change for the following reasons:

Values	ChangeReason	Description
0x0001	AlarmStateChanged	The "State" property has changed
0x0003	RaisedEvent	Status change "Incoming"
0x0005	ClearEvent	Status change "Reset"
0x0007	AcknowledgeEvent	Status change "Acknowledged"
0x0009	ResetEventReason	Status change "Deleted"
0x000F	RemoveEvent	Status change "Removed"
0x0010	AlarmQualityChanged	The "Quality" property has changed
0x0020	AlarmParameterValueChanged	A value of the "AlarmParameterValues" property has changed
0x0040	AlarmPriorityChanged	The "Priority" property has changed
0x0080	AlarmSuppressionStateChanged	The "SuppressionState" property has changed
0x0100	AlarmEscalationReasonChanged	The "EscalationReason" property has changed
0x0200	AlarmEnableStateChanged	Is set together with "RemoveEvent" to show that the alarm was deactivated
0x0400	TextUpdate	Alarm texts have changed
0x1000	ConfigurationChanged	Alarm configuration has changed
0x2000	ExternalUpdate	The "ModificationTime" property was changed for certain application-specific alarm scenarios
0x8000	IgnoreInLogging	For suppressing the reporting of specific alarms.

Type

UInt16

Access

Read-only

Syntax

```
AlarmResult.ChangeReason
```

See also

AlarmResult (Page 1077)

AlarmResult.ClearTime

Description

The "ClearTime" property returns the time of the reset of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
AlarmResult.ClearTime
```

See also

AlarmResult (Page 1077)

AlarmResult.Connection

Description

The "Connection" property returns the name of the connection by which an alarm was triggered.

Type

String

Access

Read-only

Syntax

```
AlarmResult.Connection
```

See also

AlarmResult (Page 1077)

AlarmResult.Duration

Description

The "Duration" property returns the duration of an alarm.

Type

Time

Access

Read-only

Syntax

```
AlarmResult.Duration
```

See also

AlarmResult (Page 1077)

AlarmResult.EventText

Description

The "EventText" property returns a localized text that describes an alarm event.

The text can contain triggered placeholders and reference all "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".

Type

String

Access

Read-only

Syntax

```
AlarmResult.EventText
```

See also

AlarmResult (Page 1077)

AlarmResult.Flashing

Description

The "Flashing" property returns whether the specified object flashes in runtime.

Value	Status
1	Alarm flashes
0	Alarm does not flash

A background color for flashing is specified with the "BackColor" property. The second background color and the frequency of the flashing is configured in the alarm control.

Type

Bool

Access

Read-only

Syntax

```
AlarmResult.Flashing
```

See also

AlarmResult (Page 1077)

AlarmResult.ID

Description

The "ID" property returns the ID of an alarm.

Type

UInt32

Access

Read-only

Syntax

```
AlarmResult.ID
```

See also

AlarmResult (Page 1077)

AlarmResult.InstanceID

Description

The "InstanceID" property returns the ID of alarms with multiple instances.

Type

UInt32

Access

Read-only

Syntax

```
AlarmResult.InstanceID
```

See also

AlarmResult (Page 1077)

AlarmResult.InvalidFlags

Description

The "InvalidFlags" property returns the cause of invalid data of an alarm.

An invalid alarm is marked with the following bits:

Bit number	InvalidFlags	Description
Bit 0	Invalid configuration flag	Alarm configuration is invalid. HMI device does not match data source.
Bit 1	Invalid timestamp flag	Data source transfers invalid time stamps.
Bit 2	Invalid alarm parameter flag	Data source transfers invalid parameter values.
Bit 3	Invalid event text flag	Runtime cannot format the text due to missing parameter values.

The valid alarm has the following properties:

- InvalidFlags = 0
- Quality = "good"

Type

UInt8

Access

Read-only

Syntax

```
AlarmResult.InvalidFlags
```

See also

AlarmResult (Page 1077)

AlarmResult.LoopInAlarm

Description

The "LoopInAlarm" property returns the name of the function that navigates from the display of an active alarm to its origin.

The required parameters of the function are returned with the associated "LoopInAlarmParameterValues" property.

Type

String

Access

Read-only

Syntax

```
AlarmResult.LoopInAlarm
```

See also

AlarmResult (Page 1077)

AlarmResult.LoopInAlarmParameterValues

Description

The "LoopInAlarmParameterValues" property returns the parameters of the function that navigates from the display of an active alarm to its origin. The associated "LoopInAlarm" property contains the name of the function used for the call, for example, the "OpenScreen" function if the origin of the alarm is a screen.

Type

Variant[]

Access

Read-only

Syntax

```
AlarmResult.LoopInAlarmParameterValues
```

See also

AlarmResult (Page 1077)

AlarmResult.ModificationTime

Description

The "ModificationTime" property returns the time stamp of the last modification of the alarm state.

The reason for the change is included in the "ChangeReason" property.

Type

DateTime

Access

Read-only

Syntax

```
AlarmResult.ModificationTime
```

See also

AlarmResult (Page 1077)

AlarmResult.Name

Description

The "Name" property returns the name of the alarm ("Alarm" object).

Type

String, HMIAlarm

Access

Read-only

Syntax

```
AlarmResult.Name
```

See also

AlarmResult (Page 1077)

AlarmResult.NotificationReason

Description

The "NotificationReason" property returns the reason for an alarm.

The property can have the following values:

- 0: Unknown (for example, if the alarm was read out from a log).
- 1: Add
- 2: Change
- 3: Remove

Type

UInt8

Access

Read-only

Syntax

```
AlarmResult.NotificationReason
```

See also

AlarmResult (Page 1077)

AlarmResult.Origin

Description

The "Origin" property returns the origin of an alarm.

For example, the "Origin" property contains system name, data source or CPU ID. You can sort and filter alarms through the "Origin" context.

The "Origin" property can be configured and, together with the "Area" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

Type

String

Access

Read-only

Syntax

```
AlarmResult.Origin
```

See also

AlarmResult (Page 1077)

AlarmResult.Priority

Description

The "Priority" property defines the relevance of an alarm or a machine status.

Type

UInt8

Access

Read-only

Syntax

```
AlarmResult.Priority
```

See also

AlarmResult (Page 1077)

AlarmResult.RaiseTime

Description

The "RaiseTime" property returns the trigger time of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
AlarmResult.RaiseTime
```

See also

AlarmResult (Page 1077)

AlarmResult.ResetTime

Description

The "ResetTime" property returns the time of the reset of an alarm. After resetting, the alarm is removed from the alarm system.

Type

DateTime

Access

Read-only

Syntax

```
AlarmResult.ResetTime
```

See also

AlarmResult (Page 1077)

AlarmResult.SourceID

Description

The "SourceID" property returns the source at which an alarm was triggered.

The value depends on the origin of the alarm and is assigned by the data source, for example, Range-ID, controller/connection name or computer name.

Type

String

Access

Read-only

Syntax

```
AlarmResult.SourceID
```

See also

AlarmResult (Page 1077)

AlarmResult.State

Description

The "State" property returns the state of an alarm.

The table below shows the possible states of an alarm.

Value	State	Description
0x00	Normal (Idle)	Not an active alarm
0x01	Raised	Incoming
0x02	RaisedCleared	Incoming and reset
0x05	RaisedAcknowledged	Incoming and acknowledged
0x06	RaisedAcknowledgedCleared	Incoming, acknowledged and reset
0x07	RaisedClearedAcknowledged	Incoming, reset and acknowledged
0x80	Removed	Alarm has been removed and is no longer available

Type

UInt32

Access

Read-only

Syntax

```
AlarmResult.State
```

See also

AlarmResult (Page 1077)

AlarmResult.StateText

Description

The "StateText" property returns the alarm state as text, for example, "Incoming" or "Outgoing".

The texts can be assigned system-wide for each alarm status.

Type

String

Access

Read-only

Syntax

```
AlarmResult.StateText
```

See also

AlarmResult (Page 1077)

AlarmResult.SuppressionState

Description

The "SuppressionState" property returns the status of visibility of an active alarm.

Value	SuppressionState	Description
0x0	Unsuppressed	Alarm is visible.
0x1	Suppressed	Alarm is configured as not visible.
0x3	Shelved	Alarm was hidden manually. The methods "Unshelve" and "Shelve" can be applied.

Type

UInt8

Access

Read-only

Syntax

```
AlarmResult.SuppressionState
```

See also

AlarmResult (Page 1077)

AlarmResult.SystemSeverity

Description

The "SystemSeverity" property returns the level of the severity of a system fault as property of a system alarm. The value of the "SystemSeverity" property also influences runtime monitoring (SystemHealthIndex).

The "SystemSeverity" property can indicate the following severity:

Value	SystemSeverity	Description
0	None	No effect on system monitoring.
1	Lowest severity	Fault with lowest impact on system monitoring.
2	Low severity	Fault with low impact on system monitoring.
3	Medium severity	Fault with medium impact on system monitoring.
4	High severity	Fault with great impact on system monitoring.
5	Highest severity	Fault with greatest impact on system monitoring.

Type

UInt16

Access

Read-only

Syntax

```
AlarmResult.SystemSeverity
```

See also

AlarmResult (Page 1077)

AlarmResult.TextColor

Description

The "TextColor" property returns the text color of the alarm state. Each alarm state has its own visual representation.

Type

UInt32

Access

Read-only

Syntax

```
AlarmResult.TextColor
```

See also

AlarmResult (Page 1077)

AlarmResult.UserName

Description

The "UserName" property returns the name of the user who triggered the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmResult.UserName
```

See also

AlarmResult (Page 1077)

AlarmResult.UserResponse

Description

The "UserResponse" property returns the expected or required user response to an alarm:

Value	UserResponse	Description
0x0	No response	Active alarm does not expect user response
0x1	Acknowledgment	Active alarm expects acknowledgment (also in group)
0x2	Reset	Active alarm expects reset (also in group)
0x5	Single acknowledgment	Active alarm explicitly expects single acknowledgment
0x6	Single reset	Active alarm explicitly expects single reset

Type

UInt16

Access

Read-only

Syntax

```
Alarmesult.UserResponse
```

See also

AlarmResult (Page 1077)

AlarmResult.Value

Description

The "Value" property returns the process value of an alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmResult.Value
```

See also

AlarmResult (Page 1077)

AlarmResult.ValueLimit

Description

The "ValueLimit" property returns the limit of the process value of an alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmResult.ValueLimit
```

See also

AlarmResult (Page 1077)

AlarmResult.ValueQuality

Description

The "ValueQuality" property returns the level of the quality of the process value of an alarm.

Type

UInt16

Access

Access depends on the object.

Syntax

```
AlarmResult.ValueQuality
```

See also

AlarmResult (Page 1077)

SysFct

Description

The "SysFct" object ("HMIAlarmingSysFct" type) enables access to the system functions of the "Alarming" object.

Object type

HMIAlarmingSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **CreateOperatorInputInformation()**
  Creates an alarm of the "OperatorInputInformation" class.
- **CreateSystemAlarm()**
  Creates an alarm of the "SystemAlarm" class.
- **CreateSystemInformation()**
  Creates an alarm of the "SystemInformation" class.

See also

Alarming (Page 1046)

SysFct.CreateOperatorInputInformation()

Description

The "CreateOperatorInputInformation" method creates an alarm of the "OperatorInputInformation" class.

Syntax

```
HMIRuntime.Alarming.SysFct.CreateOperatorInputInformation(AlarmText[
,Area][,AlarmParameterValue1][,AlarmParameterValue2]
[,AlarmParameterValue3][, AlarmParameterValue4]
[,AlarmParameterValue5][,AlarmParameterValue6]
[,AlarmParameterValue7]);
```

Parameters

AlarmText

Type: String

Specifies the alarm text.

In a script, the alarm text can be either a fixed string or a reference to a text list. Alarm texts can also be defined language-dependent using text lists.

Area

Optional, type: String

Specifies the scope of the alarm.

AlarmParameterValue1

Optional, type: Variant, Scalar

Value of the first alarm parameter.

AlarmParameterValue2

Optional, type: Variant, Scalar

Value of the second alarm parameter.

AlarmParameterValue3

Optional, type: Variant, Scalar

Value of the third alarm parameter.

AlarmParameterValue4

Optional, type: Variant, Scalar

Value of the fourth alarm parameter.

AlarmParameterValue5

Optional, type: Variant, Scalar

Value of the fifth alarm parameter.

AlarmParameterValue6

Optional, type: Variant, Scalar

Value of the sixth alarm parameter.

AlarmParameterValue7

Optional, type: Variant, Scalar

Value of the seventh alarm parameter.

Return value

ErrorCode

See also

SysFct (Page 1101)

AlarmResult.AlarmParameterValues (Page 1082)

SysFct.CreateSystemAlarm()

Description

The "CreateSystemAlarm" method creates an alarm of the "SystemAlarm" class.

Syntax

```
HMIRuntime.Alarming.SysFct.CreateSystemAlarm(AlarmText[,Area]
[,AlarmParameterValue1][,AlarmParameterValue2]
[,AlarmParameterValue3][,AlarmParameterValue4]
[,AlarmParameterValue5][,AlarmParameterValue6]
[,AlarmParameterValue7]);
```

Parameters

AlarmText

Type: String

Specifies the alarm text.

In a script, the alarm text can be either a fixed string or a reference to a text list. Alarm texts can also be defined language-dependent using text lists.

Area

Optional, type: String

Specifies the scope of the alarm.

AlarmParameterValue1

Optional, type: Variant, Scalar

Value of the first alarm parameter.

AlarmParameterValue2

Optional, type: Variant, Scalar

Value of the second alarm parameter.

AlarmParameterValue3

Optional, type: Variant, Scalar

Value of the third alarm parameter.

AlarmParameterValue4

Optional, type: Variant, Scalar

Value of the fourth alarm parameter.

AlarmParameterValue5

Optional, type: Variant, Scalar

Value of the fifth alarm parameter.

AlarmParameterValue6

Optional, type: Variant, Scalar

Value of the sixth alarm parameter.

AlarmParameterValue7

Optional, type: Variant, Scalar

Value of the seventh alarm parameter.

Return value

ErrorCode

See also

SysFct (Page 1101)

AlarmResult.AlarmParameterValues (Page 1082)

SysFct.CreateSystemInformation()

Description

The "CreateSystemInformation" method creates an alarm of the "SystemInformation" class.

The "SystemInformation" alarm class only has the state "Incoming" and is therefore only displayed in the alarm control under "Logged alarms" and not under "Pending alarms".

Syntax

```
HMIRuntime.Alarming.SysFct.CreateSystemInformation(AlarmText[,Area]
[,AlarmParameterValue1][,AlarmParameterValue2]
[,AlarmParameterValue3][,AlarmParameterValue4]
[,AlarmParameterValue5][,AlarmParameterValue6]
[,AlarmParameterValue7]);
```

Parameters

AlarmText

Type: String

Specifies the alarm text.

In a script, the alarm text can be either a fixed string or a reference to a text list. Alarm texts can also be defined language-dependent using text lists.

Area

Optional, type: String

Specifies the scope of the alarm.

AlarmParameterValue1

Optional, type: Variant, Scalar

Value of the first alarm parameter.

AlarmParameterValue2

Optional, type: Variant, Scalar

Value of the second alarm parameter.

AlarmParameterValue3

Optional, type: Variant, Scalar

Value of the third alarm parameter.

AlarmParameterValue4

Optional, type: Variant, Scalar

Value of the fourth alarm parameter.

AlarmParameterValue5

Optional, type: Variant, Scalar

Value of the fifth alarm parameter.

AlarmParameterValue6

Optional, type: Variant, Scalar

Value of the sixth alarm parameter.

`AlarmParameterValue7`

Optional, type: Variant, Scalar

Value of the seventh alarm parameter.

Return value

ErrorCode

See also

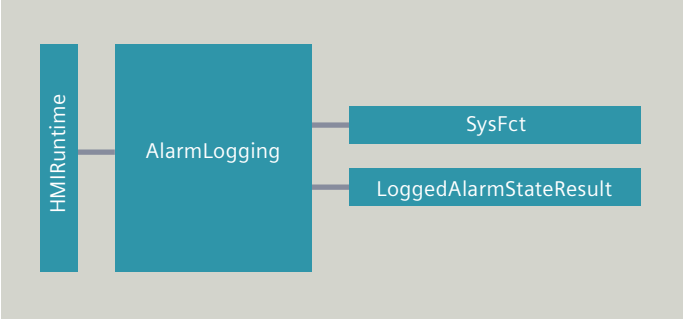
SysFct (Page 1101)

AlarmResult.AlarmParameterValues (Page 1082)

10.2.2.5 AlarmLogging

Description

The object "AlarmLogging" (type "HMIAlarmLogging") enables access to the logging system. You can read and comment on the logged alarms ("HMILoggedAlarmStateResult[]" objects) asynchronously.



Object type

HMIAlarmLogging

Properties

--

Methods

The "AlarmLogging" object has the following methods:

- **AddComment()**
  Adds a comment to logged alarms asynchronously in the logging system.
- **Read()**
  Reads out logged alarms of a time period asynchronously from a logging system.

AlarmLogging.AddComment()

Description

The "AddComment" method adds comments for logged alarms ("HMILoggedAlarmStateResult[]" objects) asynchronously in the logging system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the write operation.

Note

The "LoggedAlarmStateObjectID", "InstanceID" and "TimeStamp" parameters must correspond to the properties of the associated "HMILoggedAlarmStateResult" object.

Syntax

```
HMIRuntime.AlarmLogging.AddComment(LoggedAlarmStateObjectID,Instance
ID,TimeStamp,Language,Comment)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

LoggedAlarmStateObjectID

Type: String

ID of the logged alarm

InstanceID

Type: UInt32

InstanceID of the logged alarm

`TimeStamp`

Type: DateTime

Time stamp of the comment

`Language`

Type: UInt32, HMILCID

Country identification of the language of the comment

`Comment`

Type: String

Comment on the logged alarms

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

AlarmLogging (Page 1106)

LoggedAlarmStateResult (Page 1110)

AlarmLogging.Read()

Description

The "Read" method reads logged alarms ("HMILoggedAlarmStateResult[]" objects) of a time period asynchronously from a logging system.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern is called after the read operation with an Array of "HMILoggedAlarmStateResult" objects or an error code as parameter.

Syntax

```
HMIRuntime.AlarmLogging.Read(dateFrom,dateTo,filter,language,
[systemNames])
.then(function(alarmResultArray) {
    ...
})
```

```
.catch(function(errorCode) {
    ...
});
```

Parameters

dateFrom

Type: DateTime

End date of the time period

dateTo

Type: DateTime

Start date of the time period

filter

Type: String, HMIAlarmFilterString

SQL-type string for filtering the result set of the logged alarms

language

Type: UInt32, HMILCID

Country code of the language of the logged alarms and the filter

systemNames

Type: String[], HMISystem

Array of system names

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMILoggedAlarmStateResult[] (Page 1110) as parameter of the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

AlarmLogging (Page 1106)

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult

Description

The "LoggedAlarmStateResult" object ("HMILoggedAlarmStateResult" type) provides access to the properties of a logged alarm. The "LoggedAlarmStateResult" object is a pure data object that maps all properties of a logged alarm.

Object type

HMILoggedAlarmStateResult

Properties

The "LoggedAlarmStateResult" object has the following properties:

- **AcknowledgementTime**
  Returns the time when an alarm was acknowledged.
- **AlarmClassName**
  Returns the name of the alarm class.
- **AlarmClassSymbol**
  Returns the abbreviation for the display of the alarm class.
- **AlarmGroupID**
  Returns the alarm group ID.
- **AlarmParameterValues**
  Returns an array with parameter values of an alarm.
- **AlarmText**
  Returns the localized additional texts 1-9 of an alarm as array.
- **Area**
  Returns the area of origin of an alarm.
- **BackColor**
  Returns the background color of an alarm.
- **ChangeReason**
  Returns the event that triggered the change in the alarm state.
- **ClearTime**
  Returns the time of the reset of an alarm.
- **Connection**
  Returns the name of the connection via which an alarm was triggered.
- **Deadband**
  Returns the hysteresis value of the trigger tag of an alarm.
- **EventText**
  Returns a localized text describing an event to the alarm.
- **HostName**
  Returns the name of the PC at which the alarm was triggered.

- **ID**
  Returns the ID of a logged alarm.
- **InfoText**
  Returns the information text of an alarm in all archived languages.
- **InstanceID**
  Returns the ID of alarms with multiple instances.
- **InvalidFlags**
  Selects the valid or invalid alarm condition.
- **LoggedAlarmStateObjectID**
  Returns the ID of a logged alarm.
- **ModificationTime**
  Returns the time stamp of the last change of the alarm state.
- **Origin**
  Returns the origin of an alarm.
- **Priority**
  Returns the relevance of an alarm or a machine status.
- **RaiseTime**
  Returns the trigger time of an alarm.
- **ResetTime**
  Returns the time of the reset of an alarm.
- **SingleAcknowledgement**
  Returns whether an alarm must be acknowledged exclusively or can also be acknowledged in a group.
- **SourceType**
  Returns the type of alarm source.
- **State**
  Returns the state of an alarm.
- **StateMachine**
  Returns the behavior of the alarm for states and events.
- **StateText**
  Returns the alarm state as text.
- **SuppressionState**
  Returns the status of visibility of an active alarm.
- **TextColor**
  Returns the text color of the alarm state.
- **UserName**
  Returns the name of the user who triggered the alarm object.
- **UserResponse**
  Returns the expected or required user response to an alarm.
- **Value**
  Returns the process value of an alarm.

- **ValueLimit**
  Returns the limit of a process value of an alarm.
- **ValueQuality**
  Returns the level of the quality of a process value of an alarm.

Methods

--

See also

AlarmLogging (Page 1106)

LoggedAlarmStateResult.AcknowledgementTime

Description

The "AcknowledgementTime" property returns the time of the acknowledgment of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
LoggedAlarmStateResult.AcknowledgementTime
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.AlarmClassName

Description

The "AlarmClassName" property returns the name of the alarm class.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.AlarmClassName
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.AlarmClassSymbol

Description

The "AlarmClassSymbol" property returns the abbreviation for the display of the alarm class of the alarm, for example, "W" for the alarm class "Warning".

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.AlarmClassSymbol
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.AlarmGroupID

Description

The "AlarmGroupID" property returns the alarm group ID.

Type

UInt8

Access

Read-only

Syntax

```
LoggedAlarmStateResult.AlarmGroupID
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.AlarmParameterValues

Description

The "AlarmParameterValues" property returns an Array with parameter values of an alarm.

The parameter values are added to an alarm from the source in the "Incoming" and "Reset" state. They can also include diagnostic or raw data from the PLC in addition to simple tag values from the configured AlarmParamterTags.

Type

Variant[]

Access

Read-only

Syntax

```
LoggedAlarmStateResult.AlarmParameterValues
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.AlarmText

Description

The "AlarmText" property returns the localized additional texts 1-9 of an alarm as Array. The text can contain triggered placeholders and reference all "AlarmParameterValues" properties of the respective alarm state "Incoming" or "Reset".

Type

String[]

Access

Read-only

Syntax

```
LoggedAlarmStateResult.AlarmText
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.Area

Description

The "Area" property returns the area of origin of an alarm.

The "Area" property can be configured and, together with the "Origin" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

The "Area" property, for example, includes subsystem, application name or PLC ID. You can sort and filter alarms through the "Area" context.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.Area
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.BackColor

Description

The "BackColor" property returns the background color of an alarm.

Type

UInt32

Access

Read-only

Syntax

```
LoggedAlarmStateResult.BackColor
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ChangeReason

Description

The "ChangeReason" property returns the event that triggered the change in the alarm state. The time of last modification is saved in the "ModificationTime" property.

The alarm status may change for the following reasons:

Values	ChangeReason	Description
0x0001	AlarmStateChanged	The "State" property has changed
0x0003	RaisedEvent	Status change "Incoming"
0x0005	ClearEvent	Status change "Reset"
0x0007	AcknowledgeEvent	Status change "Acknowledged"
0x0009	ResetEventReason	Status change "Deleted"
0x000F	RemoveEvent	Status change "Removed"
0x0010	AlarmQualityChanged	The "Quality" property has changed
0x0020	AlarmParameterValueChanged	A value of the "AlarmParameterValues" property has changed
0x0040	AlarmPriorityChanged	The "Priority" property has changed
0x0080	AlarmSuppressionStateChanged	The "SuppressionState" property has changed
0x0100	AlarmEscalationReasonChanged	The "EscalationReason" property has changed

Values	ChangeReason	Description
0x0200	AlarmEnableStateChanged	Is set together with "RemoveEvent" to show that the alarm was deactivated
0x0400	TextUpdate	Alarm texts have changed
0x1000	ConfigurationChanged	Alarm configuration has changed
0x2000	ExternalUpdate	The "ModificationTime" property was changed for certain application-specific alarm scenarios
0x8000	IgnoreInLogging	For suppressing the reporting of specific alarms.

Type

UInt16

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ChangeReason
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ClearTime

Description

The "ClearTime" property returns the time of the reset of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ClearTime
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.Connection

Description

The "Connection" property returns the name of the connection by which an alarm was triggered.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.Connection
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.Deadband

Description

The "Deadband" property returns the hysteresis value of the trigger tag of an alarm.

Type

Variant

Access

Read-only

Syntax

```
LoggedAlarmStateResult.Deadband
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.EventText

Description

The "EventText" property returns a localized text that describes an alarm event.

The text can contain triggered placeholders and reference all "AlarmParameterValues" properties of the respective "incoming" or "reset" alarm state.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.EventText
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.HostName

Description

The "HostName" property returns the name of the PC at which an alarm was triggered.

Type

Boolean

Access

Read-only

Syntax

```
LoggedAlarmStateResult.HostName
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ID

Description

The "ID" property returns the ID of a logged alarm.

Type

UInt32

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ID
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.InfoText

Description

The "InfoText" property returns the information text of an alarm in all archived languages. Normally, the text is used for operator instructions.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.InfoText
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.InstanceID

Description

The "InstanceID" property returns the ID of alarms with multiple instances.

Type

UInt32

Access

Read-only

Syntax

```
LoggedAlarmStateResult.InstanceID
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.InvalidFlags

Description

The "InvalidFlags" property marks the valid or invalid alarm state.

Type

UInt8

Access

Read-only

Syntax

```
LoggedAlarmStateResult.InvalidFlags
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.LoggedAlarmStateObjectID

Description

The "LoggedAlarmStateObjectID" property returns the ID of a logged alarm.

The ID is used for referencing a logged alarm, for example, for commenting with the "AlarmLogging.AddComment" method.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.LoggedAlarmStateObjectID
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ModificationTime

Description

The "ModificationTime" property returns the time stamp of the last modification of the alarm state.

The reason for the change is included in the "ChangeReason" property.

Type

DateTime

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ModificationTime
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.Origin

Description

The "Origin" property returns the origin of an alarm.

The "Origin" property, for example, includes system names, data source or CPU ID. You can sort and filter alarms through the "Origin" context.

The "Origin" property can be configured and, together with the "Area" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.Origin
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.Priority

Description

The "Priority" property specifies the relevance of an alarm or a machine status.

Type

UInt8

Access

Read-only

Syntax

```
LoggedAlarmStateResult.Priority
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.RaiseTime

Description

The "RaiseTime" property returns the trigger time of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
LoggedAlarmStateResult.RaiseTime
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ResetTime

Description

The "ResetTime" property returns the time of the reset of an alarm.

Type

DateTime

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ResetTime
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.SingleAcknowledgement

Description

The "SingleAcknowledgement" property returns whether an alarm must be acknowledged exclusively or can also be acknowledged in a group.

Type

Boolean

Access

Read-only

Syntax

```
LoggedAlarmStateResult.SingleAcknowledgement
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.SourceType

Description

The "SourceType" property returns the type of alarm source. During resetting, the alarm is removed from the alarm system.

Type

UInt16

Access

Read-only

Syntax

```
LoggedAlarmStateResult.SourceType
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.State

Description

The "State" property returns the state of an alarm.

Type

UInt32

Access

Read-only

Syntax

```
LoggedAlarmStateResult.State
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.StateMachine

Description

The "StateMachine" property returns the behavior of an alarm for states and events.

Type

UInt8

Access

Read-only

Syntax

```
LoggedAlarmStateResult.StateMachine
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.StateText

Description

The "StateText" property returns the alarm state as text, for example, "Incoming" or "Outgoing".

The texts can be assigned system-wide for each alarm status.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.StateText
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.SuppressionState

Description

The "SuppressionState" property returns the status of visibility of an active alarm.

Type

UInt8

Access

Read-only

Syntax

```
LoggedAlarmStateResult.SuppressionState
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.TextColor

Description

The "TextColor" property returns the text color of the alarm state. Each alarm state has its own visual representation.

Type

UInt32

Access

Read-only

Syntax

```
LoggedAlarmStateResult.TextColor
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.UserName

Description

The "UserName" property returns the name of the user who triggered the alarm object.

Type

String

Access

Read-only

Syntax

```
LoggedAlarmStateResult.UserName
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.UserResponse

Description

The "UserResponse" property returns the expected or required user response to an alarm.

Type

UInt16

Access

Read-only

Syntax

```
LoggedAlarmStateResult.UserResponse
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.Value

Description

The "Value" property returns the process value of an alarm.

Type

Variant

Access

Read-only

Syntax

```
LoggedAlarmStateResult.Value
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ValueLimit

Description

The "ValueLimit" property returns the limit of the process value of an alarm.

Type

Variant

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ValueLimit
```

See also

LoggedAlarmStateResult (Page 1110)

LoggedAlarmStateResult.ValueQuality

Description

The "ValueQuality" property returns the level of the quality of the process value of an alarm.

Type

UInt16

Access

Read-only

Syntax

```
LoggedAlarmStateResult.ValueQuality
```

See also

LoggedAlarmStateResult (Page 1110)

SysFct

Description

The "SysFct" object ("HMIAlarmLoggingSysFct" type) enables access to the system functions of the "HMIAlarmLogging" object.

Object type

HMIAlarmLoggingSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **ClearAlarmLog()**
  Removes all logged alarms from the specified logging system.

See also

AlarmLogging (Page 1106)

SysFct.ClearAlarmLog()

Description

The "ClearAlarmLog" method deletes the logged alarms from the alarm log whose name was passed via the parameter. The method removes all records from the specified alarm log. All segments up to the current segment are deleted. The remaining segment is given a new start time.

Note

No automatic backup is created before the "ClearAlarmLog" method is executed.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.AlarmLogging.SysFct.ClearAlarmLog(LogName)
.then(function() {
```

```
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

LogName

Type: String, HMIAlarmLog

Name of the logging system from which the alarms are deleted.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

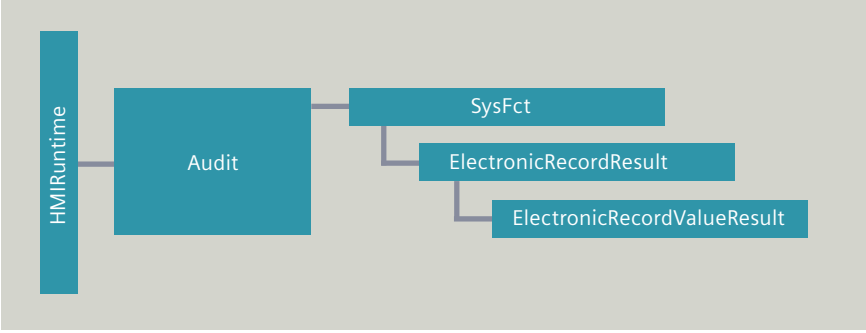
See also

SysFct (Page 1131)

AlarmLogging (Page 1106)

10.2.2.6 Audit

Description



The "Audit" object provides access to the "AuditTrail" logs contained in the project ("ElectronicRecordResult" objects) and to entries contained there ("ElectronicRecordValueResult" objects).

Object type

HMIAudit

Properties

--

Methods

--

SysFct

Description

The "SysFct" object enables access to system functions of the "Audit" object.

Type

HMIAuditSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **InsertElectronicRecord()**
  Saves an entry in the Audit Trail.
- **ReadElectronicRecord()**
  Reads entries from the Audit Trail.

SysFct.InsertElectronicRecord()

Description

The "InsertElectronicRecord" method saves an entry in the Audit Trail.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Audit.SysFct.InsertElectronicRecord(ObjectName,
Category, OperationType, OldValue, NewValue, ConfirmationType[,
Reason])
.then(function(errorCode) {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

ObjectName

Type: String

Name of the edited object.

Category

Type: String

Category that classifies the change made.

OperationType

Type: hmiOperationType

Specifies the type of change:

- hmiCreation (1): Object newly created
- hmiUpdate (2): Object edited
- hmiDeletion (3): Object deleted

OldValue

Type: Variant

Previous value of the associated object.

NewValue

Type: Variant

New value for the edited object.

ConfirmationType

Type: UInt8, hmiConfirmationType

Specifies the type of approval needed for the change:

- hmiNone (0): No acknowledgment required.
- hmiConfirmationRequired (1): Acknowledgment required.
- hmiReasonRequired (2): Acknowledgement and indication of a reason for the change is required.

Reason

Optional, type: String, HMITextList

Text list to select a reason for the change.

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  ErrorCode as parameter of the "then()" handler.
- Promise rejected (rejected):
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1133)

SysFct.ReadElectronicRecord()

Description

The "ReadElectronicRecord" method reads entries from the Audit Trail.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Audit.SysFct.ReadElectronicRecord(dateFrom, dateTo,
offSet)
.then(function(electronicRecordResult) {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

dateFrom

Type: DateTime

Start of the time period

`dateTo`

Type: DateTime

End of the time period

`offSet`

Type: UInt32

Page number of the electronic report

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIElectronicRecordResult (Page 1136) as parameter of the handler "then()".
- Promise rejected (rejected):
  ErrorCode as parameter of the "catch()" handler.

See also

ElectronicRecordResult (Page 1136)

SysFct (Page 1133)

ElectronicRecordResult

Description

The "ElectronicRecordResult" object represents an AuditTrail log.

Object type

HMIElectronicRecordResult

Properties

The "ElectronicRecordResult" object has the following properties:

- **InvalidValues**
  Specifies invalid values in the Audit Trail.
- **More**
  Specifies that the page displayed in the AuditTrail is not the last page.
- **Values**
  Specifies valid values in the Audit Trail.

Methods

--

ElectronicRecordResult.InvalidValues

Description

The "InvalidValues" property represents invalid entries ("ElectronicRecordValueResult" objects) in the Audit Trail.

Type

Object, HMIElectronicRecordValueResult[] (Page 1137)

Access

Read-write

Syntax

```
ElectronicRecordResult.InvalidValues
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult

Description

The "ElectronicRecordValueResult" object represents an entry from the AuditTrail.

Object type

HMIElectronicRecordValueResult

Properties

The "ElectronicRecordValueResult" object has the following properties:

- **AuditProvider**
  Specifies the name of the Audit Trail in which entries are stored.
- **AuditProviderType**
  Specifies the format of the AuditTrail.

- **Integrity**
  Specifies the checksum.
- **Language**
  Specifies the language of the entry.
- **NewValue**
  Specifies the new value.
- **ObjectName**
  Specifies the name of the associated object.
- **ObjectReference**
  Refers to the associated object.
- **OldValue**
  Specifies the original value.
- **OperatorStation**
  Specifies the HMI device that generates the entry.
- **OperationType**
  Defines the type of operation.
- **Reason**
  Specifies the reason for the change.
- **Signature**
  Specifies the electronic signature of the entry.
- **TimeStamp**
  Specifies the time stamp of the entry.
- **User**
  Specifies the operator who made the change.

Methods

--

ElectronicRecordValueResult.AuditProvider

Description

The "AuditProvider" property specifies the name of the Audit Trail where the entries are stored.

Type

String

Access

Read-write

Syntax

```
ElectronicRecordValueResult.AuditProvider
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.AuditProviderType

Description

The "AuditProviderType" property specifies the format of the Audit Trail.

Type

UInt16

Access

Read-write

Syntax

```
ElectronicRecordValueResult.AuditProviderType
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.Integrity

Description

The "Integrity" property specifies the checksum.

Type

UInt16

Access

Read-write

Syntax

```
ElectronicRecordValueResult.Integrity
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.Language

Description

The "Language" property specifies the language of the entry.

Type

UInt32

Access

Read-write

Syntax

```
ElectronicRecordValueResult.Language
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.NewValue

Description

The "NewValue" property specifies the new value.

Type

Variant

Access

Read-write

Syntax

```
ElectronicRecordValueResult.NewValue
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.OldValue (Page 1142)

ElectronicRecordValueResult.ObjectName

Description

The "ObjectName" property specifies the name of the associated object.

Type

Variant

Access

Read-write

Syntax

```
ElectronicRecordValueResult.ObjectName
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.ObjectReference

Description

The "ObjectReference" property refers to the associated object.

Type

Variant

Access

Read-write

Syntax

```
ElectronicRecordValueResult.ObjectReference
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.OldValue

Description

The "OldValue" property specifies the original value.

Type

Variant

Access

Read-write

Syntax

```
ElectronicRecordValueResult.OldValue
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.NewValue (Page 1140)

ElectronicRecordValueResult.OperatorStation

Description

The "OperatorStation" property specifies the HMI device that creates the entry.

Type

String

Access

Read-write

Syntax

```
ElectronicRecordValueResult.OperatorStation
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.OperationType

Description

The "OperationType" property specifies the type of the operation.

Type

UInt8, hmiOperationTyp

Specifies the type of change:

- hmiCreation (1): Entry newly created.
- hmiUpdate (2): Entry edited.
- hmiDeletion (3): Entry deleted.

Access

Read-write

Syntax

```
ElectronicRecordValueResult.OperationType
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.Reason

Description

The "Reason" property specifies the reason for the change.

Type

Variant

Access

Read-write

Syntax

```
ElectronicRecordValueResult.Reason
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.Signature

Description

The "Signature" property specifies the electronic signature of the entry.

Type

Variant

Access

Read-write

Syntax

```
ElectronicRecordValueResult.Signature
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.TimeStamp

Description

The "TimeStamp" property specifies the time stamp of the entry.

Type

DateTime

Access

Read-write

Syntax

```
ElectronicRecordValueResult.TimeStamp
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordValueResult.User

Description

The "User" property specifies the operator who made the change.

Type

String

Access

Read-write

Syntax

```
ElectronicRecordValueResult.String
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordResult.More

Description

The "More" property specifies that the displayed page in the AuditTrail is not the last page.

Type

Bool

Access

Read-write

Syntax

```
ElectronicRecordResult.More
```

See also

ElectronicRecordResult (Page 1136)

ElectronicRecordResult.Values

Description

The "Values" property represents valid entries ("ElectronicRecordValueResult" objects) in the AuditTrail.

Type

Object, HMIElectronicRecordValueResult[] (Page 1137)

Access

Read-write

Syntax

```
ElectronicRecordResult.Values
```

See also

ElectronicRecordValueResult (Page 1137)

ElectronicRecordResult (Page 1136)

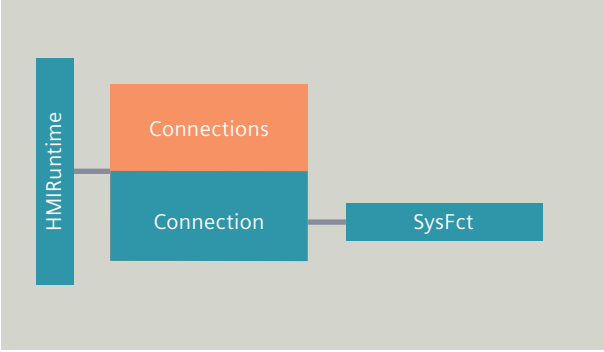
ElectronicRecordValueResult

Description

ElectronicRecordValueResult (Page 1137)

10.2.2.7 Connections

Description



The "Connections" object ("HMIConnections" type) enables access to the connections of the Runtime system. A connection is a configured, logical assignment of two communication partners.

Use

Note

The "Connections" object is not a list, but rather a "Factory". You create an instance of the "Connection" object using the tag name.

The "Connection" objects cannot be counted and enumerated like conventional lists.

To reduce the use of the "Connections" object, you can also use the alias `Connections` for `HMIRuntime.Connections`.

Object type

HMIConnections

Properties

--

Methods

The "Connections" object has the following methods:

- **Item()**
  Returns a connection ("Connection" object) of the runtime system.

Connections.Item()

Description

The "Item" method returns a connection ("Connection" object) of the runtime system.

Syntax

```
HMIRuntime.Connections[.Item](name);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Connections" object.

Parameters

`name`

Type: String, HMIConnection

Name of a connection

Return value

Object, HMIConnection (Page 1148)

See also

Connection (Page 1148)

Connection

Description

The "Connection" object ("HMIConnection" type) enables access to individual connections of the Runtime system. A connection is a configured, logical assignment of two communication partners.

Object type

HMIConnection

Properties

--

Methods

The "Connection" object has the following methods:

- **SetConnectionMode()**
  Changes the status of a connection ("Connection" object) in the Runtime system.

Connection.SetConnectionMode()

Description

The "SetConnectionMode" method changes the status of a connection ("Connection" object) in the runtime system.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
Connection.SetConnectionMode(mode)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

mode

Type: Int32, hmiConnectionMode

Specifies the connection status:

- Disabled (0): Disconnect
- Enabled (1): Set up connection

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the handler "then()".
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

Connection (Page 1148)

SysFct

Description

The "SysFct" object ("HMIConnectionsSysFct" type) enables access to the system functions of the "Connections" object.

Object type

HMIConnectionsSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **ChangeConnection()**
  Changes the connection parameters of an HMI connection in the runtime system.
- **SetConnectionMode()**
  Changes the status of a connection in the runtime system.

See also

Connection (Page 1148)

SysFct.ChangeConnection()

Description

The "ChangeConnection" method changes the connection parameters of an HMI connection.

Because the function is executed synchronously, the return value returns an error code that provides immediate information about the cause of the error. The error code can only be read if the function is called via a script.

Note

Change of function parameters after a function call

With the execution of the function, you change the function parameters. The new connection may not be active yet at this point.

Note

Usage on devices of the S7 Plus PLC family

For devices of the S7 Plus PLC family (PLCs 15xx and 12xx) it is not possible to change the slot or the rack. The system function cannot be executed if parameters for slot or rack are set.

Syntax

```
HMIRuntime.Connections.SysFct.ChangeConnection(ConnectionName,IPv4Address[,Slot][,Rack]);
```

Parameters

`ConnectionName`

Type: String, HMIConnection

Indicates the name of the connection.

`IPv4Address`

Type: String, HMIIPv4Address

Specifies the IPv4 address. Example: 192.169.153.45

`Slot`

Optional, type: UInt8

Specifies the slot number. Permitted values from 1 to 32.

`Rack`

Optional, type: UInt8

Specifies the rack number. Permitted values from 0 to 7.

Return value

ErrorCode

See also

SysFct (Page 1150)

SysFct.SetConnectionMode()

Description

The "SetConnectionMode" method changes the status of a connection ("Connection" object) in the runtime system.

Syntax

```
HMIRuntime.Connections.SysFct.SetConnectionMode(ConnectionName,Enabl
eState);
```

Parameters

ConnectionName

Type: String, HMIConnection

Name of a connection in the runtime system

EnableState

Type: Bool

Status of a connection in the runtime system:

- False: Disconnect
- True: Set up connection

Return value

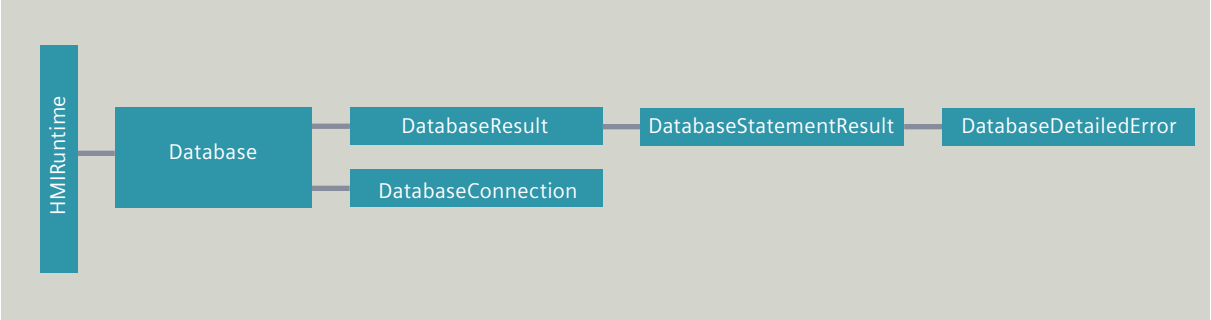
ErrorCode

See also

SysFct (Page 1150)

10.2.2.8 Database

Description



The "Database" object represents the ODBC interface. You use this interface to access the data in a database using SQL commands.

Requirement is that an ODBC interface is installed on the HMI device.

Object type

HMIDatabase

Properties

--

Methods

The "Database" object has the following methods:

- **CreateConnection()**
  Establishes the connection to a database.

Database.CreateConnection()

Description

The "CreateConnection" method establishes the connection to a database.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern with the object "DatabaseConnection" or "DatabaseResult" as parameter is called after the operation.

Syntax

```
HMIRuntime.Database.CreateConnection(connectionString)
.then(function(DatabaseConnection) {
    ...
})
.catch(function(Result) {
    ...
});
```

Parameters

connectionString

Type: String

Name of the database

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIDatabaseConnection (Page 1154) as parameter of the "then()" handler
- Promise rejected (rejected)
  Object, HMIDatabaseResult (Page 1157) as parameter of the "catch()" handler

See also

DatabaseConnection (Page 1154)

DatabaseResult (Page 1157)

Database (Page 1153)

DatabaseConnection

Description

The "DatabaseConnection" object displays the connection to a database.

Object type

HMIDatabaseConnection

Properties

--

Methods

The "DatabaseConnection" object has the following methods:

- **Close()**
  Terminates the connection to a database.
- **Execute()**
  Executes a query in a database.

See also

Database (Page 1153)

DatabaseConnection.Close()

Description

The "Close" method terminates the connection to a database.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern with the "HMIDatabaseResult" object as parameter is called after the execution.

Syntax

```
DatabaseConnection.Close()
.then(function(DatabaseResult) {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

--

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIDatabaseResult (Page 1157) as parameter of the "then()" handler
- Promise rejected (rejected)
  Object, HMIDatabaseResult (Page 1157) as parameter of the "catch()" handler.

See also

DatabaseConnection (Page 1154)

DatabaseResult (Page 1157)

DatabaseConnection.Execute()

Description

The "Execute" executes a query in the database.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern with the "HMIDatabaseResult" object as parameter is called after the execution.

Syntax

```
DatabaseConnection.Execute(query[,values])
.then(function(DatabaseResult) {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

query

Type: String

Query

values

Type: Variant | Variant[]

Value array

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIDatabaseResult (Page 1157) as parameter of the "then()" handler
- Promise rejected (rejected)
  Object, HMIDatabaseResult (Page 1157) as parameter of the "catch()" handler.

See also

DatabaseResult (Page 1157)

DatabaseConnection (Page 1154)

DatabaseResult

Description

DatabaseResult (Page 1157)

DatabaseResult

Description

The "DatabaseResult" object represents the result of a database query.

Object type

HMIDatabaseResult

Properties

The "DatabaseResult" object has the following properties:

- **GlobalError**
  Returns the global error ID.
- **Results**
  Contains the result of the database query.

Methods

--

See also

Database (Page 1153)

DatabaseResult.GlobalError

Description

The "GlobalError" property returns the global error ID.

Type

ErrorCode

Access

Read-only

Syntax

```
DatabaseResult.GlobalError
```

See also

DatabaseResult (Page 1157)

DatabaseResult.Results

Description

The "Results" property contains the result of the database query.

Type

Object[], HMIDatabaseStatementResult (Page 1159)

Access

Read-only

Syntax

```
DatabaseResult.Results
```

See also

DatabaseStatementResult (Page 1159)

DatabaseStatementResult

Description

The "DatabaseStatementResult" object represents table rows of a database query.

Object type

HMIDatabaseStatementResult

Properties

The "DatabaseStatementResult" object has the following properties:

- **Errors**
  Returns the error descriptions.
- **Rows**
  Returns the rows of the database table.

Methods

--

See also

Database (Page 1153)

DatabaseStatementResult.Errors

Description

The "Errors" property returns the error descriptions.

Type

Object[], HMIDatabaseDetailedError (Page 1160)

Access

Read-only

Syntax

```
DatabaseStatementResult.Errors
```

See also

DatabaseDetailedError (Page 1160)

DatabaseDetailedError

Description

The "DatabaseDetailedError" object displays the error description of a failed database query.

Object type

HMIDatabaseDetailedError

Properties

The "DatabaseDetailedError" object has the following properties:

- **Message**
  Returns the error description.
- **State**
  Returns the ODBC error type.

Methods

--

See also

Database (Page 1153)

DatabaseDetailedError.Message

Description

The "Message" property returns the error description.

Type

String

Access

Read-only

Syntax

```
DatabaseDetailedError.Message
```

See also

DatabaseDetailedError (Page 1160)

DatabaseDetailedError.State

Description

The "State" property returns the ODBC error type.

Type

String

Access

Read-only

Syntax

```
DatabaseDetailedError.State
```

See also

DatabaseDetailedError (Page 1160)

DatabaseStatementResult.Rows

Description

The "Rows" property returns the rows of the database table.

Type

Object[]

Access

Read-only

Syntax

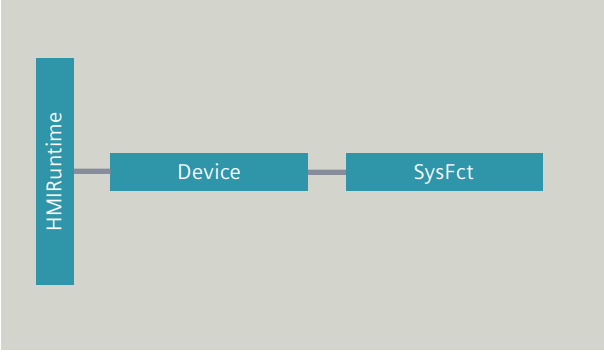
```
DatabaseStatementResult.Rows
```

See also

DatabaseStatementResult (Page 1159)

10.2.2.9 Device

Description



The "Device" object represents a device that is configured in "Devices & networks".

Object type

HMIDevice

Properties

--

Methods

--

SysFct

Description

The "SysFct" object enables access to the system functions of the "Device" object.

Object type

HMIDeviceSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **CreateScreenshot()**
  Creates a screenshot of the current screen and saves it to a specified location.
- **EjectStorageMedium()**
  Safely ejects the external storage medium.
- **GetBrightness()**
  Returns the value of the screen brightness.
- **GetDHCPState()**
  Returns the DHCP status.
- **GetIPV4Address()**
  Returns the static IPV4 address of a network adapter.
- **GetNetworkInterfaceState()**
  Returns the status of the network adapter.
- **GetSmartServerState()**
  Returns the status of the SmartServers.
- **SetBrightness()**
  Sets the brightness of the screen.
- **SetDHCPState()**
  Enables DHCP for the specified network adapter.
- **SetIPV4Address()**
  Sets the static IPV4 address of a network adapter.
- **SetNetworkInterfaceState()**
  Changes the status of the network adapter.
- **SetSmartServerState()**
  Sets the status of the SmartServers.
- **ShowControlPanel()**
  Shows the Control Panel of the HMI Panel.
- **ShowSoftwareVersion()**
  Displays the software version.
- **StartProgram()**
  Starts an external application.
- **StopRuntime()**
  Ends the Runtime and the current project.

SysFct.CreateScreenshot()

Description

The "CreateScreenshot" method creates a screenshot of the current screen and saves it to a specified location.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.CreateScreenshot(StoragePath)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

StoragePath

Type: String

Path for the location of the screenshot

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.EjectStorageMedium()

Description

The "EjectStorageMedium" method safely ejects the external storage medium.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.EjectStorageMedium(StorageDevice)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

StorageDevice

Type: String, HMIStorageDevice

External storage medium, e.g. SD-X51, USB-X61, USB-X62

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.GetBrightness()

Description

The "GetBrightness" method returns the value of the screen brightness.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.GetBrightness(Value)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`Value`

Type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the current brightness of the HMI device is written.

Note

Define the HMI tag of the "Value" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct.CreateSetTagCommand() (Page 1382)

SysFct (Page 1162)

SysFct.GetDHCPState()

Description

The "GetDHCPState" method returns the DHCP status.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful

(then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.GetDHCPState(AdapterName,State[,IPV6])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

AdapterName

Type: String

Name of the network adapter, for example, "X1"

State

Type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the DHCP status is written.

Note

Define the HMI tag of the "State" parameter with the system function "CreateSetTagCommand". Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

IPV6

Optional, type: Bool

Type of IP address:

- True: IPV6 address
- False or undefined: IPV4 address

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct.CreateSetTagCommand() (Page 1382)

SysFct (Page 1162)

SysFct.GetIPV4Address()

Description

The "GetIPV4Address" returns the static IP address of a network adapter.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.GetIPV4Address(AdapterName,IPAddress,Subnet
Mask[,DefaultGateway][,DNSServer1][,DNSServer2])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`AdapterName`

Type: String

Name of the network adapter, for example, "X1"

`IPAddress`

Type: Object, HMISetValueCommandBase

Specifies the HMI tag into which the IPv4 address of the network adapter is written in dotted decimal notation, e.g. 192.168.133.15.

Note

Define the HMI tag of the "IPAddess" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

`SubnetMask`

Type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the new subnet mask of the network adapter is written in dotted decimal notation, e.g. 255.255.255.0.

Note

Define the HMI tag of the "SubnetMask" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

`DefaultGateway`

Optional, type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the IP address of the default gateway is written in dotted decimal notation, e.g. 192.168.133.1.

Note

Define the HMI tag of the "DefaultGateway" parameter with the "CreateSetTagCommand" system function. Use, for example, the `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` instruction for the "MyTag" HMI tag.

`DNSServer1`

Optional, type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the IP address of the primary DNS server, for example, 192.168.133.1, is written.

Note

Define the HMI tag of the "DNSServer1" parameter with the "CreateSetTagCommand" system function. Use, for example, the `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` instruction for the "MyTag" HMI tag.

`DNSServer2`

Optional, type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the IP address of the secondary DNS server, for example, 192.168.133.2, is written.

Note

Define the HMI tag of the "DNSServer2" parameter with the "CreateSetTagCommand" system function. Use, for example, the `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` instruction for the "MyTag" HMI tag.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.GetNetworkInterfaceState()

Description

The "GetNetworkInterfaceState" method returns the status of the network adapter.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.GetNetworkInterfaceState(AdapterName,State)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

AdapterName

Type: String

Name of the network adapter, for example, "X1"

State

Type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the status of the network adapter is written.

Note

Define the HMI tag of the "State" parameter with the system function "CreateSetTagCommand". Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct.CreateSetTagCommand() (Page 1382)

SysFct (Page 1162)

SysFct.GetSmartServerState()

Description

The "GetSmartServerState" method returns the status of the SmartServer.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.GetSmartServerState(State)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`State`

Type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the SmartServer status is written.

- True: activated
- False: deactivated

Note

Define the HMI tag of the "State" parameter with the system function "CreateSetTagCommand". Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct.CreateSetTagCommand() (Page 1382)

SysFct.SetBrightness()

Description

The "SetBrightness" method specifies the brightness of the screen.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.SetBrightness(Value)
.then(function() {
    ...
})
.catch(function(errorCode) {
```

```
    ...
});
```

Parameters

Value

Type: UInt16

Brightness in percent

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.SetDHCPState()

Description

The "SetDHCPState" enables DHCP for the specified network adapter.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.SetDHCPState(AdapterName,Enabled[,IPV6])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`AdapterName`

Type: String

Name of the network adapter, for example, "X1"

`Enabled`

Type: Bool

DHCP mode

`IPV6`

Optional, type: Bool

Type of IP address:

- True: IPV6 address
- False or undefined: IPV4 address

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.SetIPV4Address()

Description

The "SetIPV4Address" method sets the static IP address of a network adapter.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.SetIPV4Address(AdapterName,IPAddress,Subnet
Mask[,DefaultGateway][,DNSServer1][,DNSServer2])
.then(function() {
```

```
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

AdapterName

Type: String

Name of the network adapter, for example, "X1"

IPAddress

Type: Object, HMISetValueCommandBase

IP4 Address of the network adapter in dotted decimal notation, e.g. 192.168.133.15

SubnetMask

Type: Object, HMISetValueCommandBase

New subnet mask of the network adapter in dotted decimal notation, e.g. 255.255.255.0

DefaultGateway

Optional, type: Object, HMISetValueCommandBase

IP address of the default gateway in dotted decimal notation, e.g. 192.168.133.1

DNSServer1

Optional, type: Object, HMISetValueCommandBase

IP address of the primary DNS server, e.g. 192.168.133.1

DNSServer2

Optional, type: Object, HMISetValueCommandBase

IP address of the secondary DNS server, e.g. 192.168.133.2

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.SetNetworkInterfaceState()

Description

The "SetNetworkInterfaceState" method sets the status of the network adapter.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.SetNetworkInterfaceState(AdapterName,Enable
d)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

AdapterName

Type: String

Name of the network adapter, for example, "X1"

Enabled

Type: Bool

Status of the network adapter

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.SetSmartServerState()

Description

The "SetSmartServerState" method sets the status of the SmartServer.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.SetSmartServerState(State)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

State

Type: Bool

SmartServer status:

- True: activated
- False: deactivated

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.ShowControlPanel()

Description

The "ShowControlPanel" method shows the Control Panel of the HMI Panel.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.ShowControlPanel(StartPage)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

StartPage

Type: String

Home page of the Control Panel of the Panel.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.ShowSoftwareVersion()

Description

The "ShowSoftwareVersion" method shows the software version of the runtime.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.ShowSoftwareVersion()
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

--

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

SysFct.StartProgram()

Description

The "StartProgram" method starts an external application.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.StartProgram(ProgramName,ProgramParams,Disp
layMode,WaitForProgramToEnd[,Result])
.then(function() {
```

```
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`ProgramName`

Type: String

Name of the application

`ProgramParams`

Type: String

Parameters of the application

`DisplayMode`

Type: UInt16

Display mode of the application

`WaitForProgramToEnd`

Type: Bool

Continues to execute the function list or waits until the application is terminated.

`Result`

Optional, type: Object, HMISetValueCommandBase

Specifies the HMI tag to which the result of the external application is written.

Note

Define the HMI tag of the "Result" parameter with the system function "CreateSetTagCommand". Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct.CreateSetTagCommand() (Page 1382)

SysFct (Page 1162)

SysFct.StopRuntime()

Description

The "StopRuntime" method terminates the Runtime and the project that is running.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Device.SysFct.StopRuntime([Mode])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

Mode

Optional, type: HMIStopRuntimeMode

Specifies the type of termination:

- hmiStopRuntime (0): Ends the Runtime.
- hmiStopRuntimeAndRebootDevice (1): Ends the Runtime and restarts the device.

Return value

Promise

Depending on the state of the Promise object:

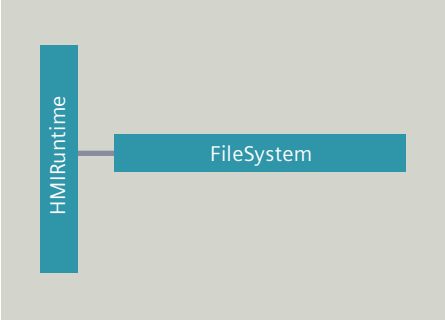
- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1162)

10.2.2.10 FileSystem

Description



The "FileSystem" object ("HMIFileSystem" type) enables access to the file system of the server on which WinCC Unified is installed.

Object type

HMIFileSystem

Properties

--

Methods

The "FileSystem" object has the following methods:

- **AppendFile()**
  Appends text to the end of a text file in the file system.
- **AppendFileBinary()**
  Appends binary data to the end of a binary file in the file system.
- **Browse()**
  Lists files and directories of a selected directory.
- **CreateDirectory()**
  Creates a new directory in the file system.
- **DeleteDirectory()**
  Deletes a directory with all subdirectories and files contained in the file system.
- **DeleteFile()**
  Deletes a file from the file system.
- **GetSpecialFolder()**
  Returns the path of a special directory.
- **IsDirectory()**
  Checks if the path name is a directory in the file system.

- **ReadFile()**
  Reads the contents of a text file from the file system.
- **ReadFileBinary()**
  Reads the contents of a binary file from the file system.
- **WriteFile()**
  Writes text to a new file in the file system.
- **WriteFileBinary()**
  Writes binary data to a new file in the file system.

FileSystem.AppendFile()

Description

The "AppendFile" method appends text to the end of a text file in the file system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
HMIRuntime.FileSystem.AppendFile(path,data,encoding)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the text file in the file system

data

Type: String

Content that is appended to the text file.

encoding

Type: String

Encoding of text file, e.g. UFT-8 or UCS-2.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.AppendFileBinary() (Page 1184)

FileSystem.AppendFileBinary()

Description

The "AppendFileBinary" method appends binary data to the end of a binary file in the file system.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, after the execution, the corresponding handler of the Promise pattern is called.

Syntax

```
HMIRuntime.FileSystem.AppendFileBinary(path,data)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the binary file in the file system

data

Type: Blob

Content that is appended to the binary file.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.AppendFile() (Page 1183)

FileSystem.Browse()

Description

The "Browse" method lists files and directories of a selected directory.

This method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Note

Browsing directories with many files and subdirectories can lead to system performance degradation until the result is available. Use the parameter `filter` to limit the search.

Syntax

```
HMIRuntime.FileSystem.Browse(path[ ,filter][ ,recursive])
.then(function(data) {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`path`

Type: String

Path of the directory in the file system

`filter`

Optional, type: String

Filters for file names. Does not filter any directory names.

Wildcards are allowed:

- "*": Any sequence of characters (can be empty)
- "?": Exactly one random character.

`recursive`

Optional, type: Bool

- True: Search all included directories.
- False: Search only the selected directory.

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  String[], as parameter of the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

Example

List all ".txt" files and directories from "/Temp" recursively:

Copy code

```
let fs = HMIRuntime.FileSystem;
fs.Browse("/Temp", "*.txt", true)
.then(/*handle path names*/);
```

See also

FileSystem (Page 1182)

FileSystem.CreateDirectory()

Description

The "CreateDirectory" method creates a new directory in the file system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
HMIRuntime.FileSystem.CreateDirectory(path)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the directory in the file system

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.DeleteDirectory() (Page 1187)

FileSystem.DeleteDirectory()

Description

The "DeleteDirectory" method deletes a directory with all subdirectories and files contained in the file system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
HMIRuntime.FileSystem.DeleteDirectory(path)
.then(function() {
    ...
```

```
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the directory in the file system

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.CreateDirectory() (Page 1186)

FileSystem.DeleteFile()

Description

The "DeleteFile" method deletes a file in the file system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
HMIRuntime.FileSystem.DeleteFile(path)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the file in the file system

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.GetSpecialFolder()

Description

The "GetSpecialFolder" method returns the path of a special directory.

The following paths are typical:

Type	Path in Windows	Path in Linux
Temporary folder	C:/WINDOWS/ServiceProfiles/WCCILScs-Service /AppData/Local/Temp/	/var/tmp/
Home directory of the user	C:/WINDOWS/ServiceProfiles/WCCILScs-Service/	/home/industrial/

Syntax

```
HMIRuntime.FileSystem.GetSpecialFolder(Folder);
```

Parameters

Folder

Type: Int32, FolderId

Specifies the special directory:

- TempDir (0): Special directory for the temporary files
- HomeDir (1): Special directory for the files of the current user

Return value

String

Example

Copy code

```
let tempDir =
HMIRuntime.FileSystem.GetSpecialFolder(HMIRuntime.FileSystem.Enums.FolderId.TempDir);
```

See also

FileSystem (Page 1182)

FileSystem.IsDirectory()

Description

The "IsDirectory" method checks whether the path name is a directory in the file system.

Note

The returned path names of the `FileSystem.Browse` method can be passed directly to this method.

Syntax

```
HMIRuntime.FileSystem.IsDirectory(path)
```

Parameters

`path`

Type: String

Path of the object in the file system

Return value

Bool

Example

List all ".txt" files and directories from "/Temp" and check whether they are directories or files:

Copy code

```
let fs = HMIRuntime.FileSystem;
fs.Browse("/Temp", "*.txt", false)
.then(function (pathnames) {
   for (let path of pathnames) {
          if (fs.IsDirectory(path)) { /*handle directory*/

      } else { /*handle file*/}
       })
.catch(function (ErrorCode) { /*handle error*/});
```

See also

FileSystem (Page 1182)

FileSystem.ReadFile()

Description

The "ReadFile" method reads the contents of a text file from the file system.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern is called with the content of the file or the error code as parameter after the execution.

Syntax

```
HMIRuntime.FileSystem.ReadFile(path,encoding)
.then(function(data) {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the text file in the file system

encoding

Type: String

Encoding of text file, e.g. UTF-8 or UCS-2.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  String as parameter of the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.ReadFileBinary() (Page 1192)

FileSystem.ReadFileBinary()

Description

The "ReadFileBinary" method reads the contents of a binary file from the file system.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern is called with the content of the file or the error code as parameter after the execution.

Syntax

```
HMIRuntime.FileSystem.ReadFileBinary(path)
.then(function(data) {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the binary file in the file system

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled
  Blob as parameter of the "then()" handler
- Promise rejected
  ErrorCode as parameter of the handler "catch()".

See also

FileSystem (Page 1182)

FileSystem.ReadFile() (Page 1191)

FileSystem.WriteFile()

Description

The "WriteFile" method writes text to a new file in the file system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
HMIRuntime.FileSystem.WriteFile(path,data,encoding)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the new text file in the file system

data

Type: String

Content that is written to the new text file.

encoding

Type: String

Encoding of the new text file, e.g. UFT-8 or UCS-2.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

FileSystem (Page 1182)

FileSystem.WriteFileBinary() (Page 1194)

FileSystem.WriteFileBinary()

Description

The "WriteFileBinary" method writes binary data to a new file in the file system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the execution.

Syntax

```
HMIRuntime.FileSystem.WriteFileBinary(path,data)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

path

Type: String

Path of the new binary file in the file system

data

Type: Blob

Content that is written to the new binary file.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

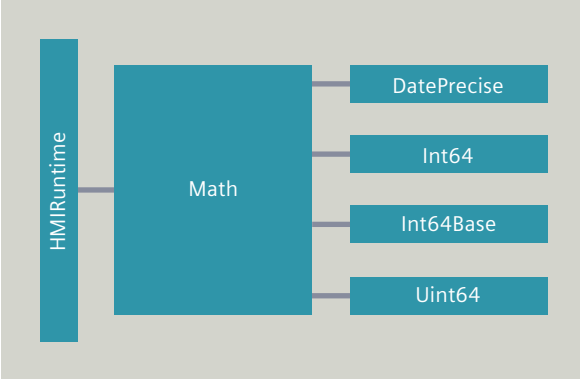
See also

FileSystem (Page 1182)

FileSystem.WriteFile() (Page 1193)

10.2.2.11 Math

Description



The "Math" object ("HMIMath" type) enables the use of 64-bit data types in the scripting environment. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as object for further processing.

Some methods return different data types; the "Tag.Read" method, for example, returns the data type "Variant". You can check the return values of these methods with the JavaScript operator "instanceof" for agreement with a 64-bit data type.

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

Object type

HMIMath

Properties

The "Math" object has the following properties:

- **DatePrecise**
  Represents a time information in 100 ns resolution.
- **Int64**
  Represents a 64-bit integer value with sign and contains methods for mathematical operations.
- **Int64Base**
  Checks for a 64-bit data type in combination with the `instanceof` operator.
- **Uint64**
  Represents a 64-bit integer value without sign and contains methods for mathematical operations.

Methods

The "Math" object has the following methods:

- **RGB()**
  Converts an RGB(A) specification into the corresponding hexadecimal value.
- **RGBWeb()**
  Converts a hexadecimal RGB(A) specification into the corresponding hexadecimal value.

Math.DatePrecise

Description

The "DatePrecise" property represents a "DatePrecise" object ("HMIDatePrecise" type) for high-resolution time information with a resolution of 100 ns as a 64-bit integer value.

The object contains methods for converting between different time-stamp formats. The following representations are supported:

- Number of milliseconds since 1970-01-01T00:00:00
- "DOMHighResTimeStamp"
  Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approx. 5 µs).
- "hrtime"
  Array of two numbers [seconds, nanoseconds]
  - seconds: Number of seconds since 1970-01-01T00:00:00Z
  - nanosecs: Number of nanoseconds that cannot be expressed in seconds (resolution is limited to 100 nanoseconds). This data type is taken from "node.js" (https://nodejs.org/api/process.html#process_process_hrtime_time).
- "fileTime"
  Number of 100 nanosecond intervals since 1601-01-01T00:00:00Z (corresponds to Win32-"FILETIME").
  This is the internal format and therefore the exact representation in terms of accuracy and value range.

The object is used, for example, to enable commenting of alarms via scripting.

Use

Note

All internal time information is available as "DatePrecise" objects. To use the native JavaScript functions for handling data objects, you must first convert the "DatePrecise" object to a JavaScript "Date" object.

Type

Object, HMIDatePrecise

Access

Read-only

Syntax

```
HMIRuntime.Math.DatePrecise;
```

Properties

--

Methods

The "DatePrecise" object has the following methods:

- **GetFiletime()**
  Returns the time information as FILETIME type.
- **GetHrTime()**
  Returns the time information as high-resolution time (hrtime) as Array in the format [seconds, nanoseconds].
- **GetMicroseconds()**
  Returns the microseconds of time information in the value range of 0 to 999.
- **GetNanoseconds()**
  Returns the nanoseconds of time information in the value range of 0 to 999.
- **GetTime()**
  Returns the time information as type "DOMHighResTimeStamp".
- **Item()**
  Creates a precise time information as a 64-bit integer value ("DatePrecise" object) and returns it.
- **SetFiletime()**
  Saves high-resolution time information as the FILETIME type.
- **SetHrTime()**
  Saves high-resolution time information of the type "hrtime".
- **SetMicroseconds()**
  Saves the microseconds of high-resolution time information.
- **SetNanoseconds()**
  Saves the microseconds of high-resolution time information.
- **SetTime()**
  Saves high-resolution time information of the type "DOMHighResTimeStamp".
- **toString()**
  Converts the time information of a "DatePrecise" object into a string.
- **valueOf()**
  Returns the time information saved in a "DatePrecise" object as "DOMHighResTimeStamp".

Example

Checks whether the time information is a "DatePrecise" object:

Copy code

```
function AlarmTriggerFunction(errorCode, SystemName, alarmResultArray) {
   let nanoSeconds; //check first if RaiseTime is a DatePrecise object
   if (alarmResultArray[0].RaiseTime instanceof HMIRuntime.Math.DatePrecise) {
      nanoSeconds = alarmResultArray[0].RaiseTime.GetNanoseconds();
   }
}
```

Converts a "DatePrecise" object to a JavaScript "Date" object so that the native JavaScript "getFullYear" function can be used:

Copy code

```
function AlarmTriggerFunction(errorCode, SystemName, alarmResultArray) {
   //convert to JavaScript-Date object first
   let fullYear = new Date(alarmResultArray[0].RaiseTime).getFullYear();
}
```

DatePrecise.GetFiletime()

Description

The "GetFiletime" method returns the time information as FILETIME type.

The type corresponds to the format of Win32-"FILETIME": Number of 100 nanosecond intervals since 1601-01-01T00:00:00Z

Syntax

```
DatePrecise.GetFiletime()
```

Parameters

--

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Math.DatePrecise (Page 1196)

DatePrecise.GetHrTime()

Description

The "GetHrTime" method returns the time information as high-resolution date (hrtime) as Array in the format [seconds, nanoseconds].

Example: 2020-10-04 11:30:22.5454118 is returned as [13238335822, 545411800].

Syntax

```
DatePrecise.GetHrTime()
```

Parameters

--

Return value

Float[], hrtime

See also

Math.DatePrecise (Page 1196)

DatePrecise.GetMicroseconds()

Description

The "GetMicroseconds" method returns the microseconds of time information in the value range of 0 to 999.

Syntax

```
DatePrecise.GetMicroseconds()
```

Parameters

--

Return value

UInt16

See also

Math.DatePrecise (Page 1196)

DatePrecise.GetNanoseconds()

Description

The "GetNanoseconds" method returns the nanoseconds of time information in the value range of 0 to 999.

Syntax

```
DatePrecise.GetNanoseconds()
```

Parameters

--

Return value

UInt16

See also

Math.DatePrecise (Page 1196)

DatePrecise.GetTime()

Description

The "GetTime" method returns the time information as type "DOMHighResTimeStamp".

The type corresponds to the following format: Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approx. 5 µs).

Syntax

```
DatePrecise.GetTime()
```

Parameters

--

Return value

Float, DOMHighResTimeStamp

See also

Math.DatePrecise (Page 1196)

DatePrecise.Item()

Description

The "Item" method creates precise time information as a 64-bit integer value ("DatePrecise" object) and returns this.

Syntax

```
DatePrecise[.Item]([year,][month,][day,][hours,][seconds,]
[milliseconds,][microseconds,][nanoseconds])
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "DatePrecise" object.

Parameters

`year`

Optional, type: Variant, HMIInt64 | Variant, HMIDatePrecise | Variant, DOMHighResTimeStamp | Variant, hrtime

Full year of the time information.

`month`

Optional, type: UInt8

Number that represents a month. (0 = January ... 11 = December)

`day`

Optional, type: UInt8

Number that represents a day (1 ... 31)

`hours`

Optional, type: UInt8

Number that represents an hour (0 ... 23)

`minutes`

Optional, type: UInt8

Number that represents a minute (0 ... 59)

`seconds`

Optional, type: UInt8

Number that represents a second (0 ... 59)

`milliseconds`

Optional, type: UInt16

Number that represents a millisecond (0 ... 999)

`microseconds`

Optional, type: UInt16

Number that represents a microsecond (0 ... 999)

nanoseconds

Optional, type: UInt16

Number that represents a nanosecond (0 ... 999)

Return value

Object, HMIDatePrecise (Page 1196)

See also

Math.DatePrecise (Page 1196)

DatePrecise.SetFiletime()

Description

The "SetFiletime" method saves high-resolution time information as type FILETIME.

Syntax

```
DatePrecise.SetFiletime([fileTime])
```

Parameters

fileTime

Optional, type: Object, HMIInt64

Time information in the format of Win32 "FILETIME": Number of 100 nanosecond intervals since 1601-01-01T00:00:00Z

Return value

--

See also

Math.DatePrecise (Page 1196)

DatePrecise.SetHrTime()

Description

The "SetHrTime" method saves high-resolution time information of type "hrtime".

Syntax

```
DatePrecise.SetHrTime(hrtime)
```

Parameters

`hrtime`

Type: Float[], hrtime

Time information of type "hrtime" as an array in format [seconds, nanoseconds]: for example, 2020-10-04 11:30:22.5454118 as [13238335822, 545411800]. The type is also returned by the "GetHrTime" method.

Return value

--

See also

Math.DatePrecise (Page 1196)

DatePrecise.SetMicroseconds()

Description

The "SetMicroseconds" method saves the microseconds of high-resolution time information.

Syntax

```
DatePrecise.SetMicroseconds(microSeconds)
```

Parameters

`microSeconds`

Type: UInt16

Microseconds of time information in the value range of 0 ... 999

Return value

--

See also

Math.DatePrecise (Page 1196)

DatePrecise.SetNanoseconds()

Description

The "SetNanoseconds" method saves the nanoseconds of a high-resolution time information.

Syntax

```
DatePrecise.SetNanoseconds(nanoSeconds)
```

Parameters

nanoSeconds

Type: UInt16

Nanoseconds of time information in the value range of 0 to 999

Return value

--

See also

Math.DatePrecise (Page 1196)

DatePrecise.SetTime()

Description

The "SetTime" method saves high-resolution time information of type "DOMHighResTimeStamp".

Syntax

```
DatePrecise.SetTime(time)
```

Parameters

time

Type: Float, DOMHighResTimeStamp

Time information: Number of milliseconds since 1970-01-01T00:00:00Z and the fraction for microseconds (resolution is approx. 5 µs). The type is also returned by the "GetTime" method.

Return value

--

See also

Math.DatePrecise (Page 1196)

DatePrecise.toString()

Description

The "toString" method converts the time information of a "DatePrecise" object into a character string.

The fixed format of the character string is "yyyy-mm-dd hh:mm:ss.HundredNanoSeconds", for example, "2020-07-04 11:30:22.5454118".

Syntax

```
DatePrecise.toString()
```

Parameters

--

Return value

String

See also

Math.DatePrecise (Page 1196)

DatePrecise.valueOf()

Description

The "valueOf" method returns the time information saved in a "DatePrecise" object as "DOMHighResTimeStamp".

The value can represent either a specific point in time (in milliseconds since 01/01/1970) or the difference between two points in time.

The value is specified in the unit milliseconds. The accuracy is up to 5 µs.

Syntax

```
DatePrecise.valueOf()
```

Parameters

--

Return value

Float, DOMHighResTimeStamp

See also

Math.DatePrecise (Page 1196)

Int64

Description

Int64 (Page 1207)

Math.Int64

Description

The "Int64" property represents a "Int64" object ("HMIInt64" type) for a 64-bit signed integer value and contains basic arithmetic and bit operations. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as "Int64" object.

Use

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

Type

Object, HMIInt64

Access

Read-only

Syntax

```
HMIRuntime.Math.Int64;
```

Properties

The "Int64" object has the following properties:

- **Hi**
  Saves and returns the high part (32-bit) of a 64-bit integer value.
- **Lo**
  Saves and returns the low part (32-bit) of a 64-bit integer value.

Methods

The "Int64" object has the following methods:

- **Add()**
  Provides the "Addition" arithmetic operation for 64-bit objects.
- **And()**
  Provides the "AND" bit operation for 64-bit objects.
- **Div()**
  Provides the "Division" arithmetic operation for 64-bit objects.
- **Item()**
  Creates 64-bit integer values ("Int64" and "Uint64" objects) and returns them.
- **Mul()**
  Provides the "Multiplication" arithmetic operation for 64-bit objects.
- **Or()**
  Provides the "OR" bit operation for 64-bit objects.
- **ShiftLeft()**
  Provides a bit shift "SHL" for 64-bit objects.
- **ShiftRight()**
  Provides the bit shift "SHR" for 64-bit objects.
- **Sub()**
  Provides the "Subtraction" arithmetic operation for 64-bit objects.
- **toString()**
  Converts the value of a 64-bit object into a string.
- **Xor()**
  Provides the "XOR" bit operation for 64-bit objects.

Example

Creates a new "signed" 64-bit object and writes the value to the tag:

Copy code

```
function Write_Int64TagValue() {
    //create new Int64-object
    var newTagVal = HMIRuntime.Math.Int64('-6000000000000000000');
    //write to tag
    HMIRuntime.Tags('Tag1').Write(newTagVal);
}
```

Checks whether the value of a "Tag" object is a signed 64-bit data type, multiplies the value with -1 and returns the result as 64-bit object:

Copy code

```
function NegMul_Int64TagValue() {
   var tagVal = HMIRuntime.Tags('Tag1').Read();
   //check if it is *signed* 64-Bit type
   if (tagVal instanceof HMIRuntime.Math.Int64) {
   //if yes, use Mul method with negative number
      return tagVal.Mul(-1);
   }
}
```

Int64.Hi

Description

The "Hi" property saves and returns the high part (32-bit) of a 64-bit integer value.

Type

UInt32

Access

Read-only

Syntax

```
Int64.Hi
```

See also

Math.Int64 (Page 1207)

Int64.Lo

Description

The "Lo" property saves and returns the low part (32-bit) of a 64-bit integer value.

Type

UInt32

Access

Read-only

Syntax

```
Int64.Lo
```

See also

Math.Int64 (Page 1207)

Int64.Add()

Description

The "Add" method provides the "Addition" arithmetic operation for 64-bit objects. The value of the "Int64" objects is increased by the specified value.

This method corresponds to the JavaScript operator "+" for other data types.

Syntax

```
Int64.Add(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value that is added to the current value of the object.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.And()

Description

The "And" method provides the "AND" bit operation for 64-bit objects. The binary value of the "Int64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "&" for other data types.

Syntax

```
Int64.And(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Bit sequence of the same length with which the binary value of the object is ANDed.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.Div()

Description

The "Div" method provides the "Division" arithmetic operation for 64-bit objects. The value of the "Int64" objects is divided by the specified value.

This method corresponds to the JavaScript operator "/" for other data types.

Syntax

```
Int64.Div(value)
```

Parameters

`value`

Type: Variant, HMIInt64 | Variant, HMIUint64

Value by which the current value of the object is divided.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.Item()

Description

The "Item" method creates 64-bit integer values ("Int64" objects) and returns them.

Syntax

```
Int64[.Item](value)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Int64" object.

Parameters

value

Type: Variant

New 64-bit integer value as integer string with base 10.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.Mul()

Description

The "Mul" method provides the "Multiplication" arithmetic operation for 64-bit objects. The value of the "Int64" objects is multiplied by the specified value.

This method corresponds to the JavaScript operator "*" for other data types.

Syntax

```
Int64.Mul(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value by which the current value of the object is multiplied.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.Or()

Description

The "Or" method provides the "OR" bit operation for 64-bit objects. The binary value of the "Int64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "|" for other data types.

Syntax

```
Int64.Or(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Bit sequence of the same length with which the binary value of the object is ORed.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.ShiftLeft()

Description

The "ShiftLeft" method provides a bit shift "SHL" for 64-bit objects. The binary value of the "Int64" objects is shifted by the specified number of digits.

This method corresponds to the JavaScript operator "<<" for other data types.

Syntax

```
Int64.ShiftLeft(value)
```

Parameters

value

Type: UInt8

Number of digits by which the binary value of the object is shifted to the left.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.ShiftRight()

Description

The "ShiftRight" method provides the bit shift "SHR" for 64-bit objects. The binary value of the "Int64" objects is shifted by the specified number of digits.

This method corresponds to the JavaScript operator ">>" for other data types.

Syntax

```
Int64.ShiftRight(value)
```

Parameters

value

Type: UInt8

Number of digits by which the binary value of the object is shifted to the right.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.Sub()

Description

The "Sub" method provides the "Subtraction" arithmetic operation for 64-bit objects. The value of the "Int64" objects is decreased by the specified value.

This method corresponds to the JavaScript operator "-" for other data types.

Syntax

```
Int64.Sub(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value that is subtracted from the current value of the object.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Int64.toString()

Description

The "toString" method converts the value of a 64-bit object into a character string. For other data types, you can use the native JavaScript method "toString" with the same name.

Syntax

```
Int64.toString([base])
```

Parameters

base

Optional, type: UInt8

Base to which 64-bit value of the object is converted to a character string. Without parameters, the basis is set to "10".

Return value

String

See also

Math.Int64 (Page 1207)

Int64.Xor()

Description

The "Xor" method provides the "XOR" bit operation for 64-bit objects. The binary value of the "Int64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "^" for other data types.

Syntax

```
Int64.Xor(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Bit sequence of the same length with which the binary value of the object is XORed.

Return value

Object, HMIInt64 (Page 1207)

See also

Math.Int64 (Page 1207)

Math.Int64Base

Description

The "Int64Base" property represents an "Int64Base" object ("HMIInt64Base" type). The object is used exclusively to check for a 64-bit integer data type. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as "Int64" or "Uint64" objects for further use.

Use

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

The "Int64Base" object returns TRUE for all signed or unsigned 64-bit integer values when checked with the "instanceof" operator against an object.

Type

Object, HMIInt64Base

Access

Read-only

Syntax

```
HMIRuntime.Math.Int64Base;
```

Properties

--

Methods

--

Example

Checks whether the value of a "Tag" object is a 64-bit data type:

Copy code

```
function Int64TagValue() {
   var tagVal = HMIRuntime.Tags('Tag1').Read(); //check if it is 64-Bit type (signed or
unsigned)
   if (tagVal instanceof HMIRuntime.Math.Int64Base) {
      ...
   }
}
```

See also

Math (Page 1195)

Math.Uint64

Description

The "Uint64" property represents a "Uint64" object ("HMIUint64" type) for a 64-bit unsigned integer value and contains basic arithmetic and bit operations. Because JavaScript does not offer native support for 64-bit integer values, these values are encapsulated as "Uint64" object.

Use

Note

64-bit values should only be processed with the methods of the "Int64" and "Uint64" objects. When you are using the native computing operations of JavaScript (e.g. "+" or "-"), the accuracy of the results is diminished.

Type

Object, HMIUint64

Access

Read-only

Syntax

```
HMIRuntime.Math.Uint64;
```

Properties

The "Uint64" object has the following properties:

- **Hi**
  Saves and returns the high part (32-bit) of a 64-bit integer value.
- **Lo**
  Saves and returns the low part (32-bit) of a 64-bit integer value.

Methods

The "Int64" object has the following methods:

- **Add()**
  Provides the "Addition" arithmetic operation for 64-bit objects.
- **And()**
  Provides the "AND" bit operation for 64-bit objects.
- **Div()**
  Provides the "Division" arithmetic operation for 64-bit objects.
- **Item()**
  Creates 64-bit integer values ("Int64" and "Uint64" objects) and returns them.
- **Mul()**
  Provides the "Multiplication" arithmetic operation for 64-bit objects.
- **Or()**
  Provides the "OR" bit operation for 64-bit objects.
- **ShiftLeft()**
  Provides a bit shift "SHL" for 64-bit objects.
- **ShiftRight()**
  Provides the bit shift "SHR" for 64-bit objects.
- **Sub()**
  Provides the "Subtraction" arithmetic operation for 64-bit objects.
- **toString()**
  Converts the value of a 64-bit object into a string.
- **Xor()**
  Provides the "XOR" bit operation for 64-bit objects.

Example

Creates a new "unsigned" 64-bit object and writes the value to the tag:

Copy code

```
function Write_Uint64TagValue() {
   //create new Uint64-object
   var newTagVal = HMIRuntime.Math.Uint64('6000000000000000000');
   //write to tag
   HMIRuntime.Tags('Tag1').Write(newTagVal);
}
```

Copy code

Checks whether the value of an object "Tag" is a signed 64-bit data type, adds the value 99 and returns the result as a 64-bit object:

Copy code

```
function Add_Int64TagValue() {
   var tagVal = HMIRuntime.Tags('Tag1').Read();
   //check if it is 64-Bit type (unsigned)
   if (tagVal instanceof HMIRuntime.Math.Uint64) {
   //if yes, use Add method
      return tagVal.Add(99);
   }
}
```

Uint64.Hi

Description

The "Hi" property saves and returns the high part (32-bit) of a 64-bit integer value.

Type

UInt32

Access

Read-only

Syntax

```
Uint64.Hi
```

See also

Math.Uint64 (Page 1218)

Uint64.Lo

Description

The "Lo" property saves and returns the low part (32-bit) of a 64-bit integer value.

Type

UInt32

Access

Read-only

Syntax

```
Uint64.Lo
```

See also

Math.Uint64 (Page 1218)

Uint64.Add()

Description

The "Add" method provides the "Addition" arithmetic operation for 64-bit objects. The value of the "Uint64" objects is increased by the specified value.

This method corresponds to the JavaScript operator "+" for other data types.

Syntax

```
Uint64.Add(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value that is added to the current value of the object.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.And()

Description

The "And" method provides the "AND" bit operation for 64-bit objects. The binary value of the "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "&" for other data types.

Syntax

```
Uint64.And(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Bit sequence of the same length with which the binary value of the object is ANDed.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.Div()

Description

The "Div" method provides the "Division" arithmetic operation for 64-bit objects. The value of the "Uint64" objects is divided by the specified value.

This method corresponds to the JavaScript operator "/" for other data types.

Syntax

```
Uint64.Div(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value by which the current value of the object is divided.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.Item()

Description

The "Item" method creates 64-bit integer values ("Uint64" objects) and returns them.

Syntax

```
Uint64[.Item](value)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Uint64" object.

Parameters

value

Type: Variant

New 64-bit integer value as integer string with base 10.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.Mul()

Description

The "Mul" method provides the "Multiplication" arithmetic operation for 64-bit objects. The value of the "Uint64" objects is multiplied by the specified value.

This method corresponds to the JavaScript operator "*" for other data types.

Syntax

```
Uint64.Mul(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value by which the current value of the object is multiplied.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.Or()

Description

The "Or" method provides the "OR" bit operation for 64-bit objects. The binary value of the "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "|" for other data types.

Syntax

```
Uint64.Or(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Bit sequence of the same length with which the binary value of the object is ORed.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.ShiftLeft()

Description

The "ShiftLeft" method provides a bit shift "SHL" for 64-bit objects. The binary value of the "Uint64" objects is shifted by the specified number of digits.

This method corresponds to the JavaScript operator "<<" for other data types.

Syntax

```
Uint64.ShiftLeft(value)
```

Parameters

value

Type: UInt8

Number of digits by which the binary value of the object is shifted to the left.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.ShiftRight()

Description

The "ShiftRight" method provides the bit shift "SHR" for 64-bit objects. The binary value of the "Uint64" objects is shifted by the specified number of digits.

This method corresponds to the JavaScript operator ">>" for other data types.

Syntax

```
Uint64.ShiftRight(value)
```

Parameters

value

Type: UInt8

Number of digits by which the binary value of the object is shifted to the right.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.Sub()

Description

The "Sub" method provides the "Subtraction" arithmetic operation for 64-bit objects. The value of the "Uint64" objects is decreased by the specified value.

This method corresponds to the JavaScript operator "-" for other data types.

Syntax

```
Uint64.Sub(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Value that is subtracted from the current value of the object.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Uint64.toString()

Description

The "toString" method converts the value of a 64-bit object into a character string. For other data types, you can use the native JavaScript method "toString" with the same name.

Syntax

```
Uint64.toString([base])
```

Parameters

base

Optional, type: UInt8

Base to which 64-bit value of the object is converted to a character string. Without parameters, the basis is set to "10".

Return value

String

See also

Math.Uint64 (Page 1218)

Uint64.Xor()

Description

The "Xor" method provides the "XOR" bit operation for 64-bit objects. The binary value of the "Uint64" objects is ANDed with the specified value.

This method corresponds to the JavaScript operator "^" for other data types.

Syntax

```
Uint64.Xor(value)
```

Parameters

value

Type: Variant, HMIInt64 | Variant, HMIUint64

Bit sequence of the same length with which the binary value of the object is XORed.

Return value

Object, HMIUint64 (Page 1218)

See also

Math.Uint64 (Page 1218)

Math.RGB()

Description

The "RGB" method converts a RGB(A) specification into the corresponding hexadecimal value.

Syntax

```
HMIRuntime.Math.RGB(R, G, B[, A]);
```

Parameters

R

Type: UInt32

Red value

G

Type: UInt32

Green value

B

Type: UInt32

Blue value

A

Optional, type: UInt32

Alpha value (density)

Return value

UInt32

See also

Math (Page 1195)

Math.RGBWeb()

Description

The "RGBWeb" method converts a hexadecimal RGB(A) specification into the corresponding hexadecimal value.

Syntax

```
HMIRuntime.Math.RGBWeb(RGB[, A]);
```

Parameters

RGB

Type: UInt32

Red-green-blue value in hexadecimal notation

A

Optional, type: UInt32

Alpha value (density) in hexadecimal format

Return value

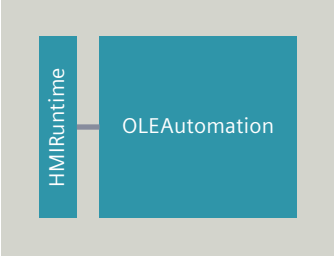
UInt32

See also

Math (Page 1195)

10.2.2.12 OLEAutomation

Description



The "OLEAutomation" object communicates in Microsoft Windows with objects of other applications (Automation objects) via script. These applications (Automation server) make their functionality available to the scripting environment (Automation client) through the COMinterface.

This gives you the opportunity to integrate WinCC Unified in existing systems, e.g. databases, Office applications or the file system. You can also integrate your own components. All COM objects that implement the IDispatch interface and have a Library type are supported.

Note

When the scripting environment is started as a service, as of Microsoft Windows Vista it is no longer possible to communicate with interactive programs (e.g. Microsoft Excel) by means of a script. Database access via ADO (ActiveX Data Objects) to a closed application is still possible.

Use

Note

Note that the scripts are always executed on the server side. Therefore all the referenced applications and files have to be available on the server.

Note

JavaScript does not support properties with parameters. Parameters must be referenced with a function type syntax for COM objects:

```
var textcell = o.Cells.Range('A1');
```

Note

JavaScript does not support functions with return parameters. Return values for COM objects must be referenced through an object:

```
var oReturnValue = {
    value: 100
};


AutomationObject.MethodWithOutParam(oReturnValue);
//out parameter set by function will be available as
"oReturnValue.value"
```

Object type

HMIOLEAutomation

Properties

--

Methods

The "OLEAutomation" object has the following methods:

- CreateObject()
  Creates a reference to an automation object.
- GetObject()
  Returns a reference to an automation object from a file.

Example

A new "testfile.txt" file with the content "this is some data" is created in the file system:

Copy code

```
function WriteToFile() {
    var fso = HMIRuntime.OLEAutomation.CreateObject('Scripting.FileSystemObject');
    var f = fso.CreateTextFile('testfile.txt');
    f.WriteLine('this is some data');
    f.Close();
}
```

The "OLEAutomation" object is only available in Microsoft Windows. This is how you check to determine if the functionality is available:

Copy code

```
if (undefined != HMIRuntime.OLEAutomation) {
    ...
}
```

OLEAutomation.CreateObject()

Description

The "CreateObject" method creates a reference to an automation object.

Syntax

```
HMIRuntime.OLEAutomation.CreateObject(progid);
```

Parameters

progid

Type: String

The name of the application that provides the object and the type or class of the object to be created.

Return value

Object

Example

Create a reference to an automation object:

Copy code

```
var obj = HMIRuntime.OLEAutomation.CreateObject('Scripting.FileSystemObject');
```

See also

OLEAutomation (Page 1229)

OLEAutomation.GetObject()

Description

The "GetObject" method returns a reference to an automation object from a file.

Syntax

```
HMIRuntime.OLEAutomation.GetObject([pathname][, class]);
```

Parameters

pathname

Optional, type: String

Full path and name of the file containing the object to be retrieved.

Note

If the "pathname" parameter is omitted, the "class" parameter is required.

class

Optional, type: String

Class of the object

Note

If the "class" parameter is omitted, the "pathname" parameter is required.

Return value

Object

Example

Return a reference to an automation object from a file:

Copy code

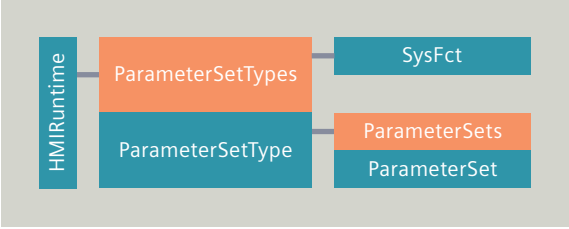
```
var obj = HMIRuntime.OLEAutomation.GetObject('C:\tmp\data.mdb');
```

See also

OLEAutomation (Page 1229)

10.2.2.13 ParameterSetTypes

Description



The "ParameterSetTypes" object represents the list of parameter set types.

To reduce the use of the "ParameterSetTypes" object, you can also use the alias `ParameterSetTypes` for `HMIRuntime.ParameterSetTypes`.

Object type

HMIParameterSetTypes

Properties

--

Methods

The "ParameterSetTypes" object has the following methods:

- **Item()**
  Returns a parameter set.

ParameterSetTypes.Item()

Description

The "Item" method returns a parameter set type ("ParameterSetType" object).

Syntax

```
[HMIRuntime.]ParameterSetTypes[.Item](parameterSetTypeId)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "ParameterSetTypes" object.

Parameters

`parameterSetTypeId`

Type: String|UInt32

Custom ID or name of the parameter set type

Return value

Object, HMIParameterSetType (Page 1234)

See also

ParameterSetTypes (Page 1233)

ParameterSetType (Page 1234)

ParameterSetType

Description

The "ParameterSetType" object represents a parameter set type.

Object type

HMIParameterSetType

Properties

The "ParameterSetType" object has the following properties:

- **ParameterSets**
  Returns the list of the parameter sets ("ParameterSets" objects).

Methods

The "ParameterSetType" object has the following methods:

- **Export()**
  Exports one or all parameter sets ("ParameterSets" objects) of a parameter set type ("ParameterSetType" object).
- **GetName()**
  Returns the name of the parameter set.
- **Import()**
  Imports one or all parameter sets ("ParameterSets" objects) of a parameter set type ("ParameterSetType" object).

See also

ParameterSetTypes (Page 1233)

ParameterSetType.Export()

Description

The "Export" method exports one or all parameter sets ("ParameterSets" objects) of a parameter set type ("ParameterSetType" object).

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
ParameterSetType.Export(ParameterSetId, FileName, OverWrite,
OutputStatus, GenerateChecksum)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetId`

Type: Variant

Custom ID or name of the parameter set.

`FileName`

Type: String

Name of the file into which the parameter set is exported.

`OverWrite`

Type: UInt32, hmiOverwrite

Specifies whether an existing parameter set is overwritten:

- Disabled (0): Do not overwrite parameter set
- Enabled (1): Overwrite parameter set

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Output status message
- False: Do not output status message

`GenerateChecksum`

Type: Bool

Specifies whether the checksum is checked during export.

- True: Generate checksum
- False: Do not generate checksum

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

ParameterSetType (Page 1234)

ParameterSetType.Import() (Page 1237)

ParameterSetType.GetName()

Description

The "GetName" method returns the name of the parameter set type.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
ParameterSetType.GetName(Language)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

Language

Type: UInt32, HMILCID

LCID of the language of the parameter set type.

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected):
  ErrorCode as parameter of the "catch()" handler.

See also

ParameterSetType (Page 1234)

ParameterSetType.Import()

Description

The "Import" method imports one or all parameter sets ("ParameterSets" objects) of a parameter set type ("ParameterSetType" object).

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
ParameterSetType.Import(FileName, ParameterSetId, OverWrite,
OutputStatus, VerifyChecksum)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`FileName`

Type: String

Name of the file from which the parameter set is imported.

`ParameterSetId`

Type: Variant

Custom ID or name of the parameter set.

`OverWrite`

Type: UInt32, hmiOverwrite

Specifies whether an existing parameter set is overwritten:

- Disabled (0): Do not overwrite parameter set
- Enabled (1): Overwrite parameter set

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Output status message
- False: Do not output status message

`VerifyChecksum`

Type: Bool

Specifies whether the checksum is checked during import.

- True: Check checksum
- False: Do not check checksum

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler. This state is only present if no alarms of the "ParameterSetType" object could be output.

See also

ParameterSetType (Page 1234)

ParameterSetType.Export() (Page 1235)

ParameterSetType.ParameterSets

Description

The "ParameterSets" property represents the list of parameter sets ("ParameterSets" objects).

Type

Object, HMIParameterSets

Access

Read-only

Syntax

```
ParameterSetType.ParameterSets
```

Properties

--

Methods

The "ParameterSets" object has the following methods:

- **Item()**
  Returns an object of the type "ParameterSet".

See also

ParameterSetType (Page 1234)

ParameterSets.Item()

Description

The "Item" method returns a parameter set ("ParameterSet" object).

Syntax

```
ParameterSets[.Item](parameterSetId)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "ParameterSets" object.

Parameters

`parameterSetId`

Type: String|UInt32

Custom ID or name of the parameter set.

Return value

Object, HMIParameterSet (Page 1240)

See also

ParameterSetType.ParameterSets (Page 1239)

ParameterSet (Page 1240)

ParameterSet

Description

The "ParameterSet" object represents a parameter set.

Object type

HMIParameterSet

Properties

--

Methods

The "ParameterSet" object has the following methods:

- **GetName()**
  Returns the name of the parameter set.
- **LoadAndWrite()**
  Loads a parameter set ("ParameterSet" object) from the memory of the HMI device and writes the parameter set to the PLC.
- **ReadAndSave()**
  Reads a parameter set ("ParameterSet" object) from the PLC and writes the parameter set to the local memory of the HMI device.

See also

ParameterSetType.ParameterSets (Page 1239)

ParameterSet.GetName()

Description

The "GetName" method returns the name of the parameter set type.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
ParameterSet.GetName(Language)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

Language

Type: UInt32, HMILCID

LCID of the language of the parameter set.

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected):
  ErrorCode as parameter of the "catch()" handler.

See also

ParameterSet (Page 1240)

ParameterSet.LoadAndWrite()

Description

The "LoadAndWrite" method loads a parameter set from the HMI device memory and writes the parameter set to the PLC.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
ParameterSet.LoadAndWrite(OutputStatus)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Output status message
- False: Do not output status message

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

ParameterSet (Page 1240)

ParameterSet.ReadAndSave() (Page 1243)

ParameterSet.ReadAndSave()

Description

The "ReadAndSave" method reads a parameter set from the PLC and writes the parameter set to the local memory of the HMI device.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
ParameterSet.ReadAndSave(OutputStatus, OverWrite)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Output status message
- False: Do not output status message

`OverWrite`

Type: UInt32, hmiOverwrite

Specifies whether an existing parameter set is overwritten:

- Disabled (0): Do not overwrite parameter set
- Enabled (1): Overwrite parameter set

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected):
  ErrorCode as parameter of the "catch()" handler.

See also

ParameterSet (Page 1240)

ParameterSet.LoadAndWrite() (Page 1242)

SysFct

Description

The "SysFct" object enables access to the system functions of the "ParameterSetTypes" object.

Object type

HMIParameterSetTypesSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **CreateParameterSet()**
  Creates a parameter set with default values configured in the Engineering System.
- **DeleteParameterSet()**
  Deletes a parameter set.
- **ExportParameterSets()**
  Exports one or all parameter sets of a parameter set type.
- **GetParameterSetName()**
  Returns the name of a parameter set based on the parameter set ID and the language.

- **GetParameterSetTypeName()**
  Returns the name of a parameter set type based on the parameter set type ID and the language.
- **ImportParameterSets()**
  Imports one or all parameter sets of a parameter set type.
- **LoadAndWriteParameterSet()**
  Loads a parameter set from the memory of the HMI and writes the parameter set to the PLC.
- **LoadParameterSet()**
  Loads a parameter set from the local memory of the HMI device and writes the parameter set to the edit tag.
- **ReadAndSaveParameterSet()**
  Reads a parameter set from the PLC and writes the parameter set to the local memory of the HMI.
- **ReadParameterSet()**
  Reads a parameter set type from the PLC to the corresponding edit tag.
- **RenameParameterSet()**
  Changes the name of a parameter set.
- **SaveParameterSet()**
  Reads a parameter set from the edit tag and writes the parameter set to the local memory of the HMI device.
- **WriteParameterSet()**
  Loads a parameter set from the edit tag and writes the parameter set to the PLC.

See also

ParameterSetTypes (Page 1233)

SysFct.CreateParameterSet()

Description

The "CreateParameterSet" method creates a parameter set with default values configured in the Engineering System.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.CreateParameterSet(ParameterSe
tType[, ParameterSetID][, ParameterSetName], OutputStatus[,
ProcessingStatus])
.then(function() {
    ...
})
```

```
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetType`

Type: Variant

Custom ID or name of the parameter set type

`ParameterSetID`

Optional, type: UInt32

Custom ID of the parameter set. If not specified, the default ID is assigned.

`ParameterSetName`

Optional, type: String

Custom name of the parameter set. If not specified, the default name is assigned.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Output status message
- False: Do not output status message

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.DeleteParameterSet()

Description

The "DeleteParameterSet" method deletes a parameter set.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.DeleteParameterSet(ParameterSe
tType, ParameterSet, OutputStatus[, ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

ParameterSetType

Type: Variant

Custom ID or name of the parameter set type

ParameterSet

Type: Variant

Custom ID or name of the parameter set.

OutputStatus

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.ExportParameterSets()

Description

The "ExportParameterSets" method exports one or all parameter sets ("ParameterSets" objects) of a parameter set type ("ParameterSetType" object).

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.ExportParameterSets(ParameterS
etTypeId, ParameterSetId, FileName, OverWrite, OutputStatus[,
ProcessingStatus], GenerateChecksum)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

ParameterSetTypeId

Type: Variant

Custom ID or name of the parameter set type

ParameterSetId

Type: Variant

Custom ID or name of the parameter set.

FileName

Type: String

Name of the file into which the parameter set is exported.

OverWrite

Type: UInt32, hmiOverwrite

Specifies whether an existing parameter set is overwritten:

- Disabled (0): Do not overwrite parameter set.
- Enabled (1): Overwrite parameter set.

OutputStatus

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

ProcessingStatus

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

`GenerateChecksum`

Type: Bool

Specifies whether the checksum is checked during export.

- True: Generate checksum.
- False: Do not generate checksum.

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.ImportParameterSets() (Page 1254)

SysFct.CreateSetTagCommand() (Page 1382)

SysFct.GetParameterSetName()

Description

The "GetParameterSetName" method returns the name of a parameter set based on the parameter set ID and the language.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then())

and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.GetParameterSetName(ParameterS
etTypeId, ParameterSetId, Language, ParameterSetName,
ProcessingStatus)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

ParameterSetTypeId

Type: UInt32

Custom ID of the parameter set type.

ParameterSetId

Type: UInt32

Custom ID of the parameter set.

Language

Type: UInt32, HMILCID

LCID of the language

ParameterSetName

Type: Object, HMISetValueCommandBase

HMI tag in which the name of the parameter set is stored.

Note

Define the HMI tag of the "ParameterSetName" parameter with the "CreateSetTagCommand" system function. Use, for example,
the `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` instruction for the "MyTag" HMI tag.

ProcessingStatus

Type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.GetParameterSetTypeName()

Description

The "GetParameterSetTypeName" method returns the name of a parameter set type based on the parameter set type ID and the language.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.GetParameterSetTypeName(Parame
terSetTypeId, Language, ParameterSetTypeName, ProcessingStatus)
.then(function() {
    ...
})
.catch(function(errorCode) {
```

```
    ...
})
```

Parameters

`ParameterSetTypeId`

Type: UInt32

Custom ID of the parameter set type.

`Language`

Type: UInt32, HMILCID

LCID of the language

`ParameterSetTypeName`

Type: Object, HMISetValueCommandBase

HMI tag in which the name of the parameter set type is stored.

Note

Define the HMI tag of the "ParameterSetTypeName" parameter with the "CreateSetTagCommand" system function. Use, for example, the `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` instruction for the "MyTag" HMI tag.

`ProcessingStatus`

Type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.ImportParameterSets()

Description

The "ImportParameterSets" method imports one or all parameter sets ("ParameterSets" objects) of a parameter set type ("ParameterSetType" object).

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.ImportParameterSets(FileName,
ParameterSetId, OverWrite, OutputStatus[, ProcessingStatus],
VerifyChecksum)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`FileName`

Type: String

Name of the file from which the parameter set is imported.

`ParameterSetId`

Type: Variant

Custom ID or name of the parameter set.

`OverWrite`

Type: UInt32, hmiOverwrite

Specifies whether an existing parameter set is overwritten:

- Disabled (0): Do not overwrite parameter set.
- Enabled (1): Overwrite parameter set.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

`VerifyChecksum`

Type: Bool

Specifies whether the checksum is checked during import.

- True: Check checksum.
- False: Do not check checksum.

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.ExportParameterSets() (Page 1248)

SysFct.CreateSetTagCommand() (Page 1382)

SysFct.LoadAndWriteParameterSet()

Description

The "LoadAndWriteParameterSet" method loads a parameter set from the HMI device memory and writes the parameter set to the PLC.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.LoadAndWriteParameterSets(Para
meterSetTypeId, ParameterSetId, OutputStatus[, ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetTypeId`

Type: Variant

Custom ID or name of the parameter set.

`ParameterSetId`

Type: Variant

Custom ID or name of the parameter set.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

ProcessingStatus

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.ReadAndSaveParameterSet() (Page 1259)

SysFct.CreateSetTagCommand() (Page 1382)

SysFct.LoadParameterSet()

Description

The "LoadParameterSet" method loads a parameter set from the local memory of the HMI device and writes the parameter set to the edit tag.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.LoadParameterSet(ParameterSetT
ype, ParameterSet, OutputStatus[, ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetType`

Type: Variant

Custom ID or name of the parameter set type

`ParameterSet`

Type: Variant

Custom ID or name of the parameter set.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.ReadAndSaveParameterSet()

Description

The "ReadAndSaveParameterSet" method reads a parameter set from the PLC and writes the parameter set to the local memory of the HMI device.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.ReadAndSaveParameterSet(Parame
terSetTypeId, ParameterSetId, OverWrite, OutputStatus[,
ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

ParameterSetTypeId

Type: Variant

Custom ID or name of the parameter set type

ParameterSetId

Type: Variant

Custom ID or name of the parameter set.

`OverWrite`

Type: UInt32, hmiOverwrite

Specifies whether an existing parameter set is overwritten:

- Disabled (0): Do not overwrite parameter set.
- Enabled (1): Overwrite parameter set.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.LoadAndWriteParameterSet() (Page 1256)

SysFct.CreateSetTagCommand() (Page 1382)

SysFct.ReadParameterSet()

Description

The "ReadParameterSet" method reads a parameter set from the PLC to the corresponding edit tag.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.ReadParameterSet(ParameterSetT
ype, OutputStatus[, ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

ParameterSetType

Type: Variant

Custom ID or name of the parameter set type

OutputStatus

Type: Bool

Specifies whether a status message is output after the transfer:

- True: Status message output.
- False: Status message not output.

ProcessingStatus

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.RenameParameterSet()

Description

The "RenameParameterSet" method changes the name of a parameter set.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.RenameParameterSet(ParameterSe
tType, ParameterSet, NewParameterSetName, OutputStatus[,
ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetType`

Type: Variant

Custom ID or name of the parameter set type

`ParameterSet`

Type: Variant

Custom ID or name of the parameter set.

`NewParameterSetName`

Type: String

New user-defined name of the parameter set.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.SaveParameterSet()

Description

The "SaveParameterSet" method reads a parameter set from the edit tag and writes the parameter set to the local memory of the HMI device.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.SaveParameterSet(ParameterSetT
ype, ParameterSet, OverWrite, OutputStatus[, ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetType`

Type: Variant

Custom ID or name of the parameter set type

`ParameterSet`

Type: Variant

Custom ID or name of the parameter set.

`OverWrite`

Type: Bool

Specifies whether an existing parameter set is overwritten:

- True: Overwrite parameter set.
- False: Do not overwrite parameter set.

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer.

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

SysFct.WriteParameterSet()

Description

The "WriteParameterSet" method loads a parameter set from the edit tag and writes the parameter set to the PLC.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]ParameterSetTypes.SysFct.WriteParameterSet(ParameterSet
Type, OutputStatus[, ProcessingStatus])
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

`ParameterSetType`

Type: Variant

Custom ID or name of the parameter set type

`OutputStatus`

Type: Bool

Specifies whether a status message is output after the transfer:

- True: Status message output.
- False: Status message not output.

`ProcessingStatus`

Optional, type: Object, HMISetValueCommandBase

HMI tag in which the processing status is stored.

Indicates the execution status of the method:

- 2: Function is being executed.
- 4: Function completed successfully.
- 12: Function was canceled.

Note

Define the HMI tag of the "ProcessingStatus" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

Return value

Promise

Depending on the state of the Promise object:

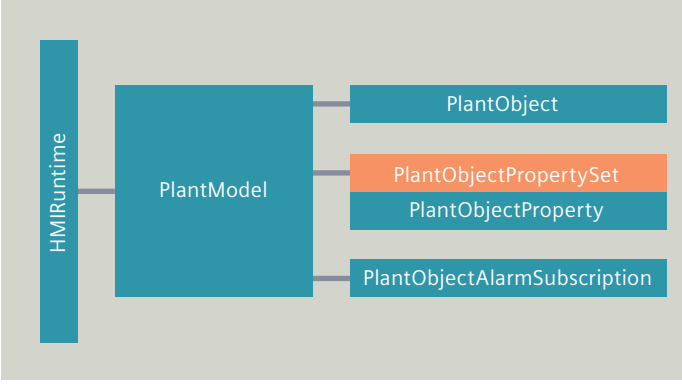
- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1244)

10.2.2.14 PlantModel

Description



The "PlantModel" object ("HMIPlantModel" type) represents the common plant model of the graphical runtime system. You reference all object instances ("PlantObject" objects) and properties of the Common Plant Models by means of the "PlantModel" object.

These object instances represent specific components or parts of the plant. This means that you have access to all the properties and methods of these objects.

Note

The Common Plant Model can map plant hierarchies in runtime. Each hierarchy consists of object instances which represent a component or a function part of the plant.

Each object instance is assigned to a hierarchy node of a plant hierarchy. Through this assignment the object instance has a unique position and address in a hierarchy.

The address in the hierarchy is represented with a path. This hierarchy path is used to reference specific objects in the properties and methods of the CommonPlantModels.

Object type

HMIPlantModel

Properties

The "PlantModel" object has the following properties:

- **LastError**
  Returns an error code for the last faulty read or write operation.

Methods

The "PlantModel" object has the following methods:

- **GetPlantObject()**
  Returns an object instance.
- **GetPlantObjectsByExpression()**
  Returns an array of object instances.
- **GetPlantObjectsByPropertyNames()**
  Returns an array of object instances.
- **GetPlantObjectsByType()**
  Returns an array of object instances.

PlantModel.LastError

Description

The "LastError" property returns an error code for the last faulty read or write operation.

Note

When the property is read out via a set, the error code of the last method call is returned.

Type

ErrorCode

Access

Read-only

Syntax

```
[HMIRuntime.]PlantModel.LastError;
```

Note

When the error occurs when accessing via a set, evaluate the "LastError" property for all objects of the set.

See also

PlantModel (Page 1267)

PlantModel.GetPlantObject()

Description

The "GetPlantObject" method returns an object instance.

Syntax

```
[HMIRuntime.]PlantModel.GetPlantObject(plantObject);
```

Parameters

plantObject

Type: String, HMIPlantObject

Name of the plant object or its position in the plant hierarchy.

Return value

Object, HMIPlantObject (Page 1273)

Example

An object instance is returned via view and path:

Copy code

```
var plantObject1 = PlantModel.GetPlantObject('.TechnologicalView::P1/S1/L2/LeftPump');
```

An object instance is returned via the name of the "HMIPlantObject" object:

Copy code

```
var plantObject2 = PlantModel.GetPlantObject('U4711');
```

See also

PlantModel (Page 1267)

PlantObject (Page 1273)

PlantModel.GetPlantObjectsByExpression()

Description

The "GetPlantObjectsByExpression" method returns an array of object instances.

Syntax

```
[HMIRuntime.]PlantModel.GetPlantObjectsByExpression(PropertyNames,Pl
antObjectType,ExpressionFilter[,ViewPath]);
```

Parameters

`PropertyNames`

Type: String, HMIPlantObjectProperty | String[], HMIPlantObjectProperty

List of the names of the properties

`PlantObjectType`

Type: String, HMIPlantObjectType

Name of the plant object type

`ExpressionFilter`

Type: String

Filter for the values of the properties.

`ViewPath`

Optional, type: String, HMIViewPath

Position of the plant object in the plant hierarchy.

Return value

Object[], HMIPlantObject[] (Page 1273)

Example

Returns an array of object instances:

Copy code

```
var plantObjectArr = PlantModel.GetPlantObjectsByExpression("Temperature", "Motor",
"Temperature>100");
```

See also

PlantModel (Page 1267)

PlantObject (Page 1273)

PlantObjectProperty (Page 1295)

PlantModel.GetPlantObjectsByPropertyNames()

Description

The "GetPlantObjectsByPropertyNames" method returns an array of object instances.

Syntax

```
[HMIRuntime.]PlantModel.GetPlantObjectsByPropertyNames(PropertyName
s[,ViewPath]);
```

Parameters

`PropertyNames`

Type: String, HMIPlantObjectProperty | String[], HMIPlantObjectProperty

List of the names of the properties

`ViewPath`

Optional, type: String, HMIViewPath

Position of the plant object in the plant hierarchy.

Return value

Object[], HMIPlantObject[] (Page 1273)

Example

Returns an array of object instances via their property names:

Copy code

```
var plantObjectArr = PlantModel.GetPlantObjects(["PPP", "ABC"]);
```

See also

PlantModel (Page 1267)

PlantObject (Page 1273)

PlantObjectProperty (Page 1295)

PlantModel.GetPlantObjectsByType()

Description

The "GetPlantObjectsByType" method returns an Array of object instances.

Syntax

```
[HMIRuntime.]PlantModel.GetPlantObjectsByType(PlantObjectType[,ViewP
ath]);
```

Parameters

`PlantObjectType`

Type: String, HMIPlantObjectType

Name of the plant object type

`ViewPath`

Optional, type: String, HMIViewPath

Position of the plant object in the plant hierarchy.

Return value

Object[], HMIPlantObject[] (Page 1273)

Example

Returns an Array of object instances via the plant object type:

Copy code

```
var plantObjectArr = PlantModel.GetPlantObjectsByType("Motor");
```

See also

PlantModel (Page 1267)

PlantObject (Page 1273)

PlantObject

Description

The "PlantObject" object ("HMIPlantObject" object) represents the object instances of the Common Plant Model.

The "PlantObject" object gives you access to all object properties or adjacent object instances in the plant hierarchy.

Note

The Common Plant Model can map any number of plant hierarchies in runtime. Each hierarchy consists of object instances which represent a component or a function part of the plant.

Each object instance is assigned to a hierarchy node in a plant hierarchy. Through this assignment the object instance has a unique position and address in a hierarchy.

The address in the hierarchy is represented with a path. This hierarchy path is used to reference specific objects in the properties and methods of the CommonPlantModels.

Object type

HMIPlantObject

Properties

The "PlantObject" object has the following properties:

- **Children**
  Returns all child object instances of an object instance in the hierarchy.
- **CurrentPlantView**
  Returns the current view of an object instance.
- **InstanceScreens**
  Returns the list of HMI screens that are assigned to an object instance.
- **LastError**
  Returns an error code for the last faulty read or write operation.
- **Name**
  Returns the object name.

- **Parent**
  Returns the higher-level object instance (Parent) that contains the current object instance as child.
- **PlantViewPaths**
  Returns the path of an object instance in a view.

Methods

The "PlantObject" object has the following methods:

- **CreateAlarmSubscription()**
  Creates a "PlantObjectAlarmSubscription" object for the object instance.
- **GetActiveAlarms()**
  Returns all active alarms of the object instance at the time of the call.
- **GetChild()**
  Returns the child object instance ("PlantObject" object) of an object instance.
- **GetProperties()**
  Returns the properties of object instances ("PlantObject" objects) as "PlantObjectPropertySet" list.

PlantObject.Children

Description

The "Children" property returns all child object instances of an object instance in the hierarchy.

Type

Object, HMIPlantObject[] (Page 1273)

Access

Read-only

Syntax

```
PlantObject.Children
```

See also

PlantObject (Page 1273)

PlantObject

Description

PlantObject (Page 1273)

PlantObject.CurrentPlantView

Description

The "CurrentPlantView" property returns the current view of an object instance. The property also specifies the view for referencing an object instance.

Type

String

Access

Read-write

Syntax

```
PlantObject.CurrentPlantView
```

Example

References the "Maintenance" view via the object path:

Copy code

```
let plantObject = PlantModel.GetPlantObject(".Maintenance::Plant1/Line3");
let maintChildren = plantObject.Children;
```

References the "Technological" view via the "CurrentPlantView" property:

Copy code

```
plantObject.CurrentPlantView = "Technological";
let techChildren = plantObject.Children;
```

See also

PlantObject (Page 1273)

PlantObject.InstanceScreens

Description

The "InstanceScreens" property returns the list of HMI screens assigned to an object instance.

Type

String[], HmiScreen[]

Access

Read-only

Syntax

```
PlantObject.InstanceScreens
```

See also

PlantObject (Page 1273)

PlantObject.LastError

Description

The "LastError" property returns an error code for the last faulty read or write operation.

Type

ErrorCode

Access

Read-only

Syntax

```
PlantObject.LastError
```

See also

PlantObject (Page 1273)

PlantObject.Name

Description

The "Name" property returns the name of an object instance.

Type

String, HMIPlantObject

Access

Read-only

Syntax

```
PlantObject.Name
```

See also

PlantObject (Page 1273)

PlantObject.Parent

Description

The "Parent" property returns the higher-level object instance (Parent) that contains the current object instance as child.

Type

Object, HMIPlantObject (Page 1273)

Access

Read-only

Syntax

```
PlantObject.Parent
```

See also

PlantObject (Page 1273)

PlantObject

Description

PlantObject (Page 1273)

PlantObject.PlantViewPaths

Description

The "PlantViewPaths" property returns the path of an object instance in a view.

Type

StringStringMap

Access

Read-only

Syntax

```
PlantObject.PlantViewPaths
```

Example

Returns the path of the object in the current view:

Copy code

```
let pathCurrent = plantObject.PlantViewPaths[plantObject.CurrentPlantView];
```

Returns the path of the object in the "Maintenance" view:

Copy code

```
let pathMaint = plantObject.PlantViewPaths["Maintenance"];
```

See also

PlantObject (Page 1273)

PlantObject.GetActiveAlarms()

Description

The "GetActiveAlarms" method returns all active alarms of the object instance at the time of the call.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "AlarmResult" objects or the error code as parameter.

Syntax

```
PlantObject.GetActiveAlarms(Language[,IncludeChildren][,Filter])
.then(function(alarmResultArray) {
    ...
});
.catch(function(errorCode) {
    ...
})
```

Parameters

`Language`

Type: UInt32, HMILCID

Language for all texts of an alarm and the filter

`IncludeChildren`

Optional, type: Bool

Specifies whether the alarms of all child object instances are included.

`Filter`

Optional, type: String, HMIAlarmFilterString

SQL-type string for filtering

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIAlarmResult[] (Page 1077) as parameter of the handler "then()".
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler. This state is only present if no alarms of the "PlantObject" object could be output.

Example

Read out all active alarms of the "PlantObject" object instance:

Copy code

```
var promise = PlantObject.GetActiveAlarms(1033);
```

See also

PlantObject (Page 1273)

AlarmResult (Page 1077)

PlantObject.GetChild()

Description

The "GetChild" method returns the child object instance ("PlantObject" object) of an object instance.

Syntax

```
PlantObject.GetChild(ChildName)
```

Parameters

`ChildName`

Type: String

Name of the child object instance in the view

Return

Object, HMIPlantObject (Page 1273)

See also

PlantObject (Page 1273)

PlantObject.GetProperties()

Description

The "GetProperties" method returns the properties of object instances ("PlantObject" objects) as "PlantObjectPropertySet" list.

Syntax

```
PlantObject.GetProperties([propertyName])
```

Parameters

propertyName

Optional, type: String | String[]

One or more properties of an object instance

Return

Object, HMIPlantObjectPropertySet (Page 1288)

Example

Returns the "Speed" and "Temperature" properties of an object instance.

Copy code

```
var plantObjectPropertySet = plantObject.GetProperties(['Speed', 'Temperature']);
```

See also

PlantObjectPropertySet (Page 1288)

PlantObject (Page 1273)

PlantObject.CreateAlarmSubscription()

Description

The "CreateAlarmSubscription" method creates a "PlantObjectAlarmSubscription" object for the object instance. With the returned object "PlantObjectAlarmSubscription" you specify the grouping of active alarms.

Syntax

```
PlantObject.CreateAlarmSubscription()
```

Parameters

--

Return value

Object, HMIPlantObjectAlarmSubscription (Page 1282)

Example

Creates an object "PlantObjectAlarmSubscription" for the composition of active alarms of the object instance "po":

Copy code

```
var po = PlantModel.GetPlantObject(".hierarchy::Node1\\Node2");
var alarmsubscription = po.CreateAlarmSubscription();
```

See also

PlantObjectAlarmSubscription (Page 1282)

PlantObject (Page 1273)

PlantObjectAlarmSubscription

Description

The "PlantObjectAlarmSubscription" object ("HMIPlantObjectAlarmSubscription" type) enables access to active alarms from all object instances of the Common Plant Models.

The object represents a selection of active alarms. The object is initialized via the "CreateAlarmSubscription" method of the "PlantObject" object. The active alarms are then grouped and called according to the properties of the "PlantObjectAlarmSubscription" object. Finally, notification is given of the changes to the alarm mapping.

Object type

HMIPlantObjectAlarmSubscription

Properties

The "PlantObjectAlarmSubscription" object has the following properties:

- **Filter**
  Defines a string for filtering active alarms.
- **IncludeChildren**
  Specifies whether all alarms of child object instances are included.
- **Language**
  Specifies the current runtime language for all texts of a message and the filter.
- **OnAlarm**
  Specifies the name of the "OnAlarm" Callback function for the monitoring of active alarms.
- **SystemNames**
  Specifies the name of the runtime system for the grouping of active alarms.

Methods

The "PlantObjectAlarmSubscription" object has the following methods:

- **Start()**
  Activates the monitoring of defined alarms of the "PlantObjectAlarmSubscription" object.
- **Stop()**
  Cancels monitoring of defined alarms of the "PlantObjectAlarmSubscription" object.

PlantObjectAlarmSubscription.Filter

Description

The "Filter" property specifies a string for filtering active alarms.

The syntax of the filter string corresponds to the WHERE clause of an SQL command.

Type

String, HMIAlarmFilterString

Access

Read-write

Syntax

```
PlantObjectAlarmSubscription.Filter
```

See also

PlantObjectAlarmSubscription (Page 1282)

AlarmSubscription.Filter (Page 1068)

PlantObjectAlarmSubscription.IncludeChildren

Description

The "IncludeChildren" property specifies whether all alarms of child object instances are included.

Type

Bool

Access

Read-write

Syntax

```
PlantObjectAlarmSubscription.IncludeChildren
```

See also

PlantObjectAlarmSubscription (Page 1282)

PlantObjectAlarmSubscription.Language

Description

The "Language" specifies the current runtime language for all texts of a message and the filter.

Type

UInt32, HMILCID

Access

Read-write

Syntax

```
PlantObjectAlarmSubscription.Language
```

See also

PlantObjectAlarmSubscription (Page 1282)

PlantObjectAlarmSubscription.OnAlarm

Description

The "OnAlarm" property specifies the name of the "OnAlarm" Callback function for the monitoring of active alarms.

The properties of the active alarms are passed to the Callback function "OnAlarm" as object "AlarmResultArray".

Required prototype of the Callback function:
`OnAlarm(errorCode,systemName,alarmResultArray)`

Type

Function, HMIOnAlarmCB

Access

Write-only

Syntax

```
PlantObjectAlarmSubscription.OnAlarm
```

Parameters

Parameters of the callback function:

errorCode

Type: ErrorCode

Error code of the active alarm

systemName

Type: String

Name of the runtime system

alarmResultArray

Type: Object, HMIAlarmResult[] (Page 1077)

Array with "AlarmResult" objects of the active alarm

See also

PlantObjectAlarmSubscription (Page 1282)

AlarmResult (Page 1077)

PlantObjectAlarmSubscription.SystemNames

Description

The "SystemNames" property specifies the name of the runtime system for compiling active alarms.

Type

String[], HMISystem

Access

Read-write

Syntax

```
PlantObjectAlarmSubscription.SystemNames
```

See also

PlantObjectAlarmSubscription (Page 1282)

PlantObjectAlarmSubscription.Start()

Description

The "Start" method activates the monitoring of defined alarms of the "PlantObjectAlarmSubscription" object.

Syntax

```
PlantObjectAlarmSubscription.Start()
```

Parameters

--

Return value

ErrorCode

Example

Start monitoring of alarms for object instance "po" and output number of alarms:

Copy code

```
var po = PlantModel.GetPlantObject(".hierarchy::Node1\\Node2");
var alarmsubscription = po.CreateAlarmSubscription(); // returns
HMIPlantObjectAlarmSubscription
alarmsubscription.Language = 1033;
alarmsubscription.OnAlarm = function(ErrorCode, SystemName, ResultSet) {

   for (let index in ResultSet) {
      HMIRuntime.Trace('Alarm Name_' + (index + 1) + ' = ' + ResultSet[index].Name);
      HMIRuntime.Trace(' Alarm State_' + (index + 1) + '= ' + ResultSet[index].State);
      HMIRuntime.Trace(' Alarm Area_' + (index + 1) + '= ' + ResultSet[index].Area);
   }
};
alarmsubscription.Filter = 'AlarmClassName=\'Alarm\'';
alarmsubscription.IncludeChildren = true;
alarmsubscription.Start();
```

See also

PlantObjectAlarmSubscription (Page 1282)

PlantObjectAlarmSubscription.Stop()

Description

The "Stop" method cancels the monitoring of defined alarms of the "PlantObjectAlarmSubscription" object.

Syntax

```
PlantObjectAlarmSubscription.Stop()
```

Parameters

--

Return value

ErrorCode

Example

Stop monitoring of alarms:

Copy code

```
var po = PlantModel.GetPlantObject(".hierarchy::Node1\\Node2");

var alarmsubscription = po.CreateAlarmSubscription(); // returns
HMIPlantObjectAlarmSubscription
alarmsubscription.Language = 1033;
alarmsubscription.OnAlarm = function(ErrorCode, SystemName, ResultSet) {
   for (let index in ResultSet) {
      HMIRuntime.Trace('Alarm Name_' + (index + 1) + ' = ' + ResultSet[index].Name);
      HMIRuntime.Trace(' Alarm State_' + (index + 1) + '= ' + ResultSet[index].State);
      HMIRuntime.Trace(' Alarm Area_' + (index + 1) + '= ' + ResultSet[index].Area);

   }
};
alarmsubscription.Filter = 'AlarmClassName=\'Alarm\'';
alarmsubscription.IncludeChildren = true;
alarmsubscription.Start();
```

See also

PlantObjectAlarmSubscription (Page 1282)

PlantObjectPropertySet

Description

The object "PlantObjectPropertySet" (type "HMIPlantObjectPropertySet") is a list "PlantObjectProperty" objects that provides optimized access to the properties of the object instances of the Common Plant Models.

After initialization of the "PlantObjectPropertySet" object, you can read and write multiple properties in a single call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

You reference the "PlantObjectPropertySet" object via the "PlantObject" object. The "PlantObjectPropertySet" object is a list and can be counted and enumerated. You can access the "PlantObjectPropertySet" list through the index or the "item" method.

Object type

HMIPlantObjectPropertySet

Properties

The "PlantObjectPropertySet" object has the following properties:

- **Count**
  Returns the number of object properties in the "PlantObjectPropertySet" list.

Methods

The "PlantObjectPropertySet" object has the following methods:

- **Add()**
  Adds one or more object properties to an existing "PlantObjectPropertySet" list.
- **Item()**
  Returns a property from the "PlantObjectPropertySet" list.
- **Read()**
  Reads in all object properties of the "PlantObjectPropertySet" list.
- **ReadAsync()**
  Asynchronously reads in all object properties of the "PlantObjectPropertySet" list.
- **Remove()**
  Removes one or more object properties from an existing "PlantObjectPropertySet" list.
- **Write()**
  Writes the values of all object properties of the "PlantObjectPropertySet" list.
- **WriteAsync()**
  Asynchronously writes the values of all object properties of the "PlantObjectPropertySet" list.

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

PlantObjectPropertySet.Count

Description

The "Count" property returns the number of object properties in the "PlantObjectPropertySet" list.

Type

UInt32

Access

Read-only

Syntax

```
PlantObjectPropertySet.Count
```

See also

PlantObjectProperty (Page 1295)

PlantObjectPropertySet (Page 1288)

PlantObjectPropertySet.Add()

Description

The "Add" method adds one or more object properties to an existing "PlantObjectPropertySet" list. The object properties are referenced by name.

Syntax

```
PlantObjectPropertySet.Add(name)
```

Parameters

name

Type: String

Name of object properties that are added to the list.

Return

Object[], HMIPlantObjectProperty[] (Page 1295)

See also

PlantObjectProperty (Page 1295)

PlantObjectPropertySet (Page 1288)

PlantObjectPropertySet.Item()

Description

The "Item" method returns a property ("PlantObjectProperty" object) from the "PlantObjectPropertySet" list.

Syntax

```
PlantObjectPropertySet[.Item](name)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "PlantObjectPropertySet" object.

Parameters

`name`

Type: String, HMIPlantObjectProperty | Int32, HMIPlantObjectProperty

Name or index number (0 ... n) of an object property of the "PlantObjectPropertySet" list.

Note

The index number of a "PlantObjectPropertySet" object does not describe the order in which the "PlantObjectProperty" objects were added to the "PlantObjectPropertySet" list.

Return

Object, HMIPlantObjectProperty (Page 1295)

See also

PlantObjectProperty (Page 1295)

PlantObjectPropertySet (Page 1288)

PlantObjectPropertySet.Read()

Description

The "Read" method reads in all object properties ("PlantObjectProperty" objects) of the "PlantObjectPropertySet" list. The value, the Quality Code and the time stamp of all properties are determined when the properties are read.

The method executes a synchronous read operation. After the read operation is complete, use the "LastError " property to determine whether the execution was successful.
if you do not want the read operation to block script execution use the method "PlantObjectPropertySet.ReadAsync".

Syntax

```
PlantObjectPropertySet.Read()
```

Parameters

--

Return

--

See also

PlantObjectPropertySet (Page 1288)

PlantObjectProperty (Page 1295)

PlantObjectPropertySet.ReadAsync()

Description

The "ReadAsync" method reads in all object properties ("PlantObjectProperty" objects) of the "PlantObjectPropertySet" list. The value, the Quality Code and the time stamp of all properties are determined when the properties are read.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, the corresponding handler of the Promise pattern with the "PlantObjectPropertySet" object or the error code as parameter is called after the operation. Execution only fails (promise rejected) when no "PlantObjectProperty" object of the "PlantObjectPropertySet" list could be read.

Syntax

```
PlantObjectProperty.ReadAsync()
.then(function(propertySet) {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

--

Return

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIPlantObjectPropertySet (Page 1288) as parameter of the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists if no "PlantObjectProperty" object of the "PlantObjectPropertySet" list could be read.

See also

PlantObjectPropertySet (Page 1288)

PlantObjectPropertySet.Remove()

Description

The "Remove" method removes one or more object properties ("PlantObjectProperty" objects) from an existing "PlantObjectPropertySet" list. The properties are referenced by the name.

Syntax

```
PlantObjectPropertySet.Remove(name)
```

Parameters

name

Type: String

Name of object properties that will be removed from the list.

Return

--

See also

PlantObjectPropertySet (Page 1288)

PlantObjectProperty (Page 1295)

PlantObjectPropertySet.Write()

Description

The "Write" method writes the values of all object properties ("PlantObjectProperty" objects) of the "PlantObjectPropertySet" list. You must first set the values of the individual object properties with the "Value" property. The value of the "Value" property does not have to correspond to the actual current value of the "PlantObjectProperty" object after the write operation is complete. If you want to update the "PlantObjectProperty" objects, execute a Read method.

The method executes a synchronous write operation. After the write operation is complete, use the "LastError " property to determine whether the execution was successful. Use the "WriteAsync" method to acquire the result of the write operation without blocking script execution,

The properties "QualityCode" and "TimeStamp" of the "PlantObjectProperty" objects are not determined during writing.

Syntax

```
PlantObjectPropertySet.Write()
```

Parameters

--

Return

--

See also

PlantObjectPropertySet (Page 1288)

PlantObjectProperty (Page 1295)

PlantObjectPropertySet.WriteAsync()

Description

The "WriteAsync" method writes the values of all object properties ("PlantObjectProperty") of the "PlantObjectPropertySet" list. You must first set the values of the individual object properties with the "Value" property. The value of the "Value" property does not have to correspond to the actual current value of the "PlantObjectProperty" objects after completion of the write operation. If you want to update the "PlantObjectProperty" objects, execute a Read method.

After the write operation is complete, use the "LastError " property to determine whether the execution was successful.

The properties "QualityCode" and "TimeStamp" of the "PlantObjectProperty" objects are not determined during writing.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "PlantObjectPropertySet" object or the error code as parameter. Execution only fails (Promise rejected) when no "PlantObjectProperty" object of the "PlantObjectPropertySet" object could be written.

Syntax

```
PlantObjectPropertySet.WriteAsync()
.then(function(propertySet) {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

--

Return

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMIPlantObjectPropertySet as parameter of the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists if no "PlantObjectProperty" object of the "PlantObjectPropertySet" list could be written.

See also

PlantObjectPropertySet (Page 1288)

PlantObjectProperty (Page 1295)

PlantObjectProperty

Description

The "PlantObjectProperty" object ("HMIPlantObjectProperty" type) represents the property of an object instance ("PlantObject" object) of the Common Plant Models.

A "PlantObjectProperty" object is returned by the "PlantObjectPropertySet" list or the `PlantObject.GetProperties` method.

Object type

HMIPlantObjectProperty

Properties

The "PlantObjectProperty" object has the following properties:

- **LastError**
  Returns an error code for the last faulty read or write operation.
- **Name**
  Returns the name of an object property.
- **QualityCode**
  Returns the level of the quality of the value of an object property after reading.
- **TimeStamp**
  Returns the time stamp of the last read operation.
- **Value**
  Specifies the value of an object property.

Methods

The "PlantObjectProperty" object has the following methods:

- **Read()**
  Reads an object property
- **Write()**
  Writes the value of an object property.

See also

PlantModel (Page 1267)

PlantObject (Page 1273)

PlantObjectPropertySet (Page 1288)

PlantObjectProperty.LastError

Description

The "LastError" property returns an error code for the last faulty read or write operation.

Type

ErrorCode

Access

Read-only

Syntax

```
PlantObjectProperty.LastError
```

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

PlantObjectProperty.Name

Description

The "Name" property returns the name of the object property.

Type

String

Access

Read-only

Syntax

```
PlantObjectProperty.Name
```

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

PlantObjectProperty.QualityCode

Description

The "QualityCode" property returns the level of the quality of the value of an object property after the read operation.

Type

UInt32

Access

Read-only

Syntax

```
PlantObjectProperty.QualityCode
```

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

Tag.QualityCode (Page 1341)

PlantObjectProperty.TimeStamp

Description

The "TimeStamp" property returns the time stamp of the last read operation.

The value 0 is returned after writing or rejected reading.

Type

DateTime

Access

Read-only

Syntax

```
PlantObjectProperty.TimeStamp
```

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

Tag.TimeStamp (Page 1345)

PlantObjectProperty.Value

Description

The "Value" property specifies a value of an object property.

Type

Variant

Access

Read-write

Syntax

```
PlantObjectProperty.Value
```

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

PlantObjectProperty.Read()

Description

The "Read" method reads a property ("PlantObjectProperty" object) of an object instance of the Common Plant Models. The value, the Quality Code and the time stamp of the object property are determined when the property is read.

The method executes a synchronous read operation. When completed, you can use the property "PlantObjectProperty.LastError" to determine if the execution was successful.

if you do not want the read operation to block script execution use the method "PlantObjectPropertySet.ReadAsync".

Syntax

```
PlantObjectProperty.Read()
```

Parameters

--

Return

Variant

See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

PlantObjectProperty.Write()

Description

The "Write" method writes the value of the property ("PlantObjectProperty" object) of an object instance of the Common Plant Models. Set the values of the individual "PlantObjectProperty" objects beforehand with the property "Value". The value of the "Value" property does not have to correspond to the actual current value of the "PlantObjectProperty" object after the write operation is complete. To update the "PlantObjectProperty" objects, execute a Read method.

The "Write" method executes a synchronous write operation. When completed, you can use the "LastError" property of the "PlantObjectProperty" object to determine if the execution was successful.

To acquire the result of the write operation without blocking script execution, use the "WriteAsync" method of the "PlantObjectPropertySet" object.

The "QualityCode" and "TimeStamp" properties of the "PlantObjectProperty" object are not determined during writing.

Syntax

```
PlantObjectProperty.Write([value])
```

Parameters

value

Optional, type: Variant

Value of the object property:

- Specify value
  The specified value overwrites the current value of the "Value" property of the "PlantObjectProperty" object.
- Without value
  The current value of the "Value" property of the "PlantObjectProperty" object is written.

Return

--

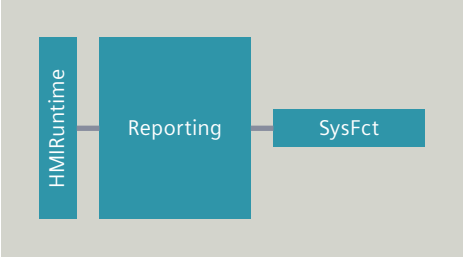
See also

PlantObjectProperty (Page 1295)

PlantObject (Page 1273)

10.2.2.15 Reporting

Description



The "Reporting" object allows access to the report function (production logs) of the runtime. When a report job is executed, the data source items defined in the report template are read from the Runtime project and their data are imported into a table in the report.

Object type

HMIReporting

Properties

--

Methods

--

SysFct

Description

The "SysFct" object enables access to the system functions of the "Reporting" object.

Object type

HMIReportingSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **ExecuteReport()**
  Starts the specified report task.

See also

Reporting (Page 1301)

SysFct.ExecuteReport()

Description

The "ExecuteReport" method starts the specified report job.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.Reporting.SysFct.Report(ReportTaskName)
.then(function() {
    ...
})
.catch(function(errorCode){
    ...
});
```

Parameters

ReportTaskName

Type: String

Name of the report job

Return value

Promise

Depending on the status of the Promise object:

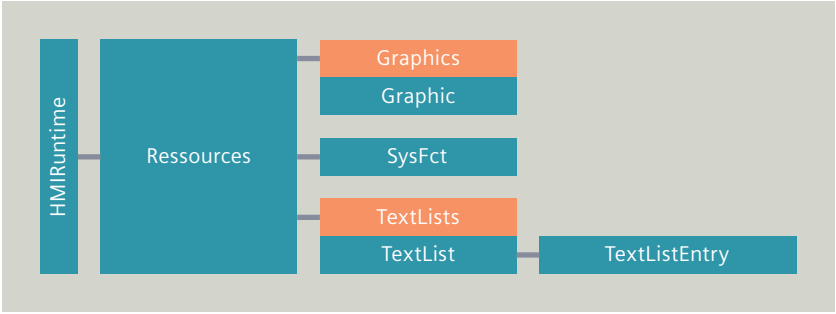
- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 1301)

10.2.2.16 Resources

Description



The "Resources" object provides access to the runtime text and graphic lists.

Object type

HMIResources

Properties

The "Resources" object has the following properties:

- **Graphics**
  Returns the "Graphics" list.
- **TextLists**
  Returns the "TextLists" list.

Methods

--

Resources.Graphics

Description

The "Graphics" property returns the list of the "Graphic" objects. In the graphic list (object "Graphics"), certain graphic ("Graphic" objects) are assigned to the possible values or value ranges of a tag.

Type

Object, HMIGraphics

Access

Read-only

Syntax

```
HMIRuntime.Resources.Graphics
```

Properties

--

Methods

The "Graphics" property has the following methods:

- **Item**
  Returns an object of the type "Graphic".

See also

Resources (Page 1303)

Graphics.Item()

Description

The "Item" method returns a "Graphic" object of the "Graphics" list.

Syntax

```
HMIRuntime.Resources.Graphics[.Item](graphicName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Graphics" object.

Parameters

`graphicName`

Type: String, HMIGraphic

Fully qualified name of the graphic, e.g. "GraphicCollection.MyImage"

Return value

Object, HMIGraphic (Page 1305)

Example

Assign a graphic to the "Button_1" button:

Copy code

```
Screen.Items("Button_1").Graphic =
HMIRuntime.Resources.Graphics("GraphicCollection.Left_Arrow").Name;
```

See also

Resources.Graphics (Page 1303)

Graphic (Page 1305)

Graphic

Description

The "Graphic" object contains the graphic assigned to a defined value or value range of a tag.

Object type

HMIGraphic

Properties

The "Graphic" object has the following properties:

- **Name**
  Returns the fully qualified name of the graphic.

Methods

--

See also

Resources.Graphics (Page 1303)

Graphic.Name

Description

The "Name" property returns the fully qualified name of the graphic, e.g. "GraphicCollection.MyImage".

Type

String

Access

Read-only

Syntax

```
Graphic.Name
```

See also

Graphic (Page 1305)

Resources.TextLists

Description

The "TextLists" property returns the list of the "TextList" objects. In the text list (object "TextLists"), certain texts ("TextList" object) are assigned in multiple languages to the possible values or value ranges of a tag.

Type

Object, HMITextLists

Access

Read-only

Syntax

```
HMIRuntime.Resources.TextLists
```

Properties

--

Methods

The "TextLists" property has the following methods:

- **Item**
  Returns an object of the type "TextList".

See also

Resources (Page 1303)

TextLists.Item()

Description

The "Item" method returns a "TextList" object of the "TextLists" list.

Syntax

```
HMIRuntime.Resources.TextLists[.Item](textListName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "TextLists" object.

Parameters

`textListName`

Type: String, HMITextList

Fully qualified name of the text list of the "TextList" object of the "TextLists" list.

Return value

Object, HMITextList (Page 1308)

See also

Resources.TextLists (Page 1306)

TextList (Page 1308)

TextList

Description

The "TextList" object contains the multilingual text list entries assigned to a defined value or value range of a tag ("TextListEntry" objects).

Object type

HMITextList

Properties

The "TextList" object has the following properties:

- **Name**
  Returns the fully qualified name of the text list.

Methods

The "TextList" object has the following methods:

- **Item()**
  Returns an entry of the text list.

See also

Resources.TextLists (Page 1306)

TextListEntry (Page 1309)

TextList.Name

Description

The "Name" property returns the fully qualified name of the text list.

Type

String

Access

Read-only

Syntax

```
TextList.Name
```

See also

TextList (Page 1308)

TextList.Item()

Description

The "Item" method returns a "TextListEntry" object of the "TextList" list.

Syntax

```
TextList[.Item](index)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "TextList" object.

Parameters

`index`

Type: Int32

Index of the "TextListEntry" object in the "TextList" list.

Return value

Object, HMITextListEntry (Page 1309)

See also

TextList (Page 1308)

TextListEntry (Page 1309)

TextListEntry

Description

The "TextListEntry" object contains the language-dependent texts of a text list.

Object type

HMITextListEntry

Properties

The "TextListEntry" object has the following properties:

- **Index**
  Returns the index of the text list entry.

Methods

The "TextListEntry" object has the following methods:

- **Item()**
  Returns a text list entry.

See also

TextList (Page 1308)

TextListEntry.Index

Description

The "Index" property returns the index of a text list entry.

Type

Int32

Access

Read-only

Syntax

```
TextListEntry.Index
```

See also

TextListEntry (Page 1309)

TextListEntry.Item()

Description

The "Item" method returns the language-dependent text of the "TextListEntry" object.

Syntax

```
TextListEntry[.Item](lcid)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "TextListEntry" object.

Parameters

`lcid`

Type: Int32, HMILCID

Language of the text

Return value

String

See also

TextListEntry (Page 1309)

SysFct

Description

The "SysFct" object enables access to the system functions of the "Resources" object.

Object type

HMIResourcesSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **LookUpText**
  Returns the contents of a text list.

See also

Resources (Page 1303)

SysFct.LookUpText()

Description

The "LookUpText" method returns the text of a text list.

Syntax

```
HMIRuntime.Resources.SysFct.LookUpText(OutputText, index, lcid,
TextListName);
```

Parameters

`OutputText`

Type: Object, HMISetValueCommandBase

HMI tag in which the selected text is stored.

Note

Define the HMI tag of the "OutputText" parameter with the "CreateSetTagCommand" system function. Use, for example, the instruction `HMIRuntime.Tags.SysFct.CreateSetTagCommand("MyTag")` for the HMI tag "MyTag".

`index`

Type: Variant

Index of the text list entry

`lcid`

Type: UInt32, HMILCID

LCID the language of the text list entry

`TextListName`

Type: String, HMITextList

Name of the text list

Return value

ErrorCode

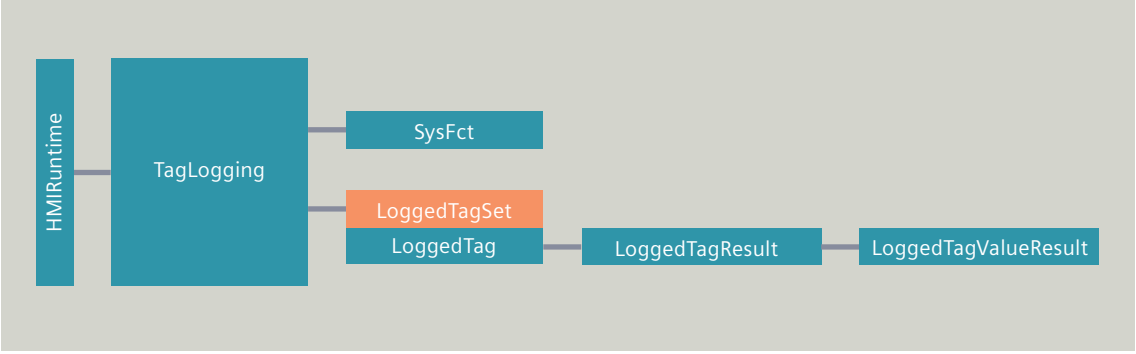
See also

SysFct (Page 1311)

SysFct.CreateSetTagCommand() (Page 1382)

10.2.2.17 TagLogging

Description



The "TagLogging" object ("HMITagLogging" type) enables access to the logging tags of a logging system.

Object type

HMITagLogging

Properties

--

Methods

The "TagLogging" object has the following methods:

- **CreateLoggedTagSet()**
  Creates a new "LoggedTagSet" object.
- **LoggedTags()**
  References a logging tag ("LoggedTag" object) of a logging system.

TagLogging.CreateLoggedTagSet()

Description

The "CreateLoggedTagSet" method creates a new "LoggedTagSet" object ("HMILoggedTagSet" type). The "LoggedTagSet" object can be filled with one or more logging tags.

You use the returned "LoggedTagSet" object for optimized read and write access to multiple logging tags.

Syntax

```
HMIRuntime.TagLogging.CreateLoggedTagSet([loggedTagNameArray]);
```

Parameter

`loggedTagNameArray`

Type: String, HMILoggedTag | String[], HMILoggedTag[]

Logging tag name or array with names of multiple logging tags that are added to the "LoggedTagSet" object. Without parameters, an empty "LoggedTagSet" object is created.

Return value

Object, HMILoggedTagSet (Page 1324)

See also

TagLogging (Page 1313)

LoggedTagSet (Page 1324)

LoggedTagSet (Page 1324)

TagLogging.LoggedTags()

Description

The "LoggedTags" method references a logging tag ("LoggedTag" object) of a logging system.

Syntax

```
HMIRuntime.TagLogging.LoggedTags(loggedTagName);
```

Parameter

`loggedTagName`

Type: String, HMILoggedTag

Logging tag name of a "LoggedTag" object

Return value

Object, HMILoggedTag (Page 1315)

See also

TagLogging (Page 1313)

LoggedTag (Page 1315)

LoggedTag

Description

The "LoggedTag" object ("HMILoggedTag" type) represents a logging tag of a logging system. A "LoggedTag" object is returned by the "TagLogging" object or the "LoggedTagSet" list.

Object type

HMILoggedTag

Properties

The "LoggedTag" object has the following property:

- **Name**
  Returns the name of the logging tag.

Methods

The "LoggedTag" object has the following methods:

- **AddComment()**
  Adds a comment to a logging tag asynchronously in the logging system.
- **Read()**
  Reads a logging tag of a period from a logging system.
- **WriteCorrectionValue()**
  Writes a correction value to a logging tag.

See also

TagLogging (Page 1313)

LoggedTagSet (Page 1324)

LoggedTag.Name

Description

The "Name" property returns the name of the logging tag ("LoggedTag" object).

Type

String, HMILoggedTag

Access

Read-only

Syntax

```
LoggedTag.Name
```

See also

LoggedTag (Page 1315)

LoggedTag.AddComment()

Description

The "AddComment" method adds a comment to a logging tag ("LoggedTag" object) asynchronously in the logging system.

The method executes an asynchronous write operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the write operation.

Note

The comment properties "User", "OperatorStation" and "CommentTimeStamp" are determined from the current script context and cannot be set manually.

Syntax

```
LoggedTag.AddComment(TimeStamp,Language,Comment)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

`TimeStamp`

Type: DateTime

Time stamp of the comment

`Language`

Type: UInt32, HMILCID

Country identification of the language of the comment

`Comment`

Type: String

Comment on the logged tags

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

LoggedTag (Page 1315)

LoggedTag.Read()

Description

The "Read" method reads out a logging tag ("LoggedTag" object) of a time period from a logging system. The value, the Quality Code, the time stamp and context information of the logging tag are determined when the logging tag is read.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, after execution is complete, the corresponding handler of the Promise pattern is called with an "LoggedTagResult" object or an error code as parameter.

Syntax

```
LoggedTag.Read(dateFrom,dateTo,boundingValue)
.then(function(loggedTagResult) {
    ...
```

```
})
.catch(function(errorCode) {
    ...
});
```

Parameters

dateFrom

Type: DateTime

End date of the time period

dateTo

Type: DateTime

Start date of the time period

boundingValue

Type: Bool

Specifies whether the limit values of the time period are transferred.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMILoggedTagResult (Page 1319) as parameter of the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

LoggedTag (Page 1315)

LoggedTagResult (Page 1319)

LoggedTag.WriteCorrectionValue()

Description

The "WriteCorrectionValue" method writes a correction value into a logging tag.

Syntax

```
LoggedTag.WriteCorrectionValue(Timestamp,Value)
```

Parameters

Timestamp

Type: DateTime

Time stamp of the logging tag whose value is being corrected.

Value

Type: Variant

Corrected value which is assigned to the logging tag.

Return value

ErrorCode

See also

LoggedTag (Page 1315)

LoggedTagResult

Description

The "LoggedTagResult" object ("HMILoggedTagResult" type) enables access to the process values of a logging tag. The "LoggedTagResult" object is returned by a read operation of objects "LoggedTag" and "LoggedTagSet" in the logging system. You have access to the process values of logging tags and errors of read operations in the logging system.

Object type

HMILoggedTagResult

Properties

The "LoggedTagResult" object has the following properties:

- **Error**
  Returns an error code for the last faulty read or write operation.
- **Name**
  Returns the name of the object or specifies it.
- **Values**
  Returns an array of process values including the quality code.

Methods

--

See also

TagLogging (Page 1313)

LoggedTag (Page 1315)

LoggedTagSet (Page 1324)

LoggedTagResult.Error

Description

The "Error" property returns an error code for the last faulty read or write operation.

Note

The value "0" is returned after a successful read or write operation. The error code always relates to the last method call of the object.

Type

ErrorCode

Access

Read-write

Syntax

```
LoggedTagResult.Error
```

See also

LoggedTagResult (Page 1319)

LoggedTagResult.Name

Description

The "Name" property returns or specifies the name of the logging tag.

Type

String, HMILoggedTag

Access

Read-write

Syntax

```
LoggedTagResult.Name
```

See also

LoggedTagResult (Page 1319)

LoggedTagResult.Values

Description

The "Values" property returns an array of process values including quality code.

Type

Object, HMILoggedTagValueResult[] (Page 1321)

Access

Read-write

Syntax

```
LoggedTagResult.Values
```

See also

LoggedTagResult (Page 1319)

LoggedTagValueResult (Page 1321)

LoggedTagValueResult

Description

The "LoggedTagValueResult" object ("HMILoggedTagValueResult" type) represents the process values of a logging tag with all associated context information.

The object is referenced via the `LoggedTagResult.Values` property.

Object type

HMILoggedTagValueResult

Properties

The "LoggedTagValueResult" object has the following properties:

- **Flags**
  Returns context information on the process value.
- **Quality**
  Returns the level of the quality of the process value.
- **TimeStamp**
  Returns the time stamp of the process value.
- **Value**
  Returns the process value.

Methods

--

See also

TagLogging (Page 1313)

LoggedTagResult.Values (Page 1321)

LoggedTagValueResult.Flags

Description

The "Flags" property returns context information on the process value.

Type

Int32, hmiTagLoggingValueFlags

Returns context information on the process value:

- Extra (0): There are still additional values at the time of the process value.
- Calculated (2): Process value is calculated.
- Bounding (16): Process value is a limit value.
- NoData (32): No additional information available
- FirstStored (64): Process value is the first value stored in the logging system.
- LastStored (128): Process value is the last value stored in the logging system.

Access

Read-write

Syntax

```
LoggedTagValueResult.Flags
```

See also

LoggedTagValueResult (Page 1321)

LoggedTagValueResult.Quality

Description

The "Quality" property returns the level of the quality of the process value.

Type

UInt32

Access

Read-write

Syntax

```
LoggedTagValueResult.Quality
```

See also

LoggedTagValueResult (Page 1321)

LoggedTagValueResult.TimeStamp

Description

The "TimeStamp" property returns the time stamp of the process value.

Type

DateTime

Access

Read-write

Syntax

```
LoggedTagValueResult.TimeStamp
```

See also

LoggedTagValueResult (Page 1321)

LoggedTagValueResult.Value

Description

The "Value" property returns the process value.

Type

Variant

Access

Read-write

Syntax

```
LoggedTagValueResult.Value
```

See also

LoggedTagValueResult (Page 1321)

LoggedTagSet

Description

The "LoggedTagSet" object ("HMILoggedTagSet" type) is a list of "LoggedTag" objects that provides optimized access to logging tags. After initialization of the "LoggedTagSet" object, you have read access to multiple logging tags in one call. Access demonstrates better performance and lower communication load than single access to multiple logging tags.

You create a new "LoggedTagSet" object with the "TagLogging.CreateLoggedTagSet" method.

The "LoggedTagSet" object can be counted and enumerated. You can access the "LoggedTagSet" list via the index or the logging tag name.

Object type

HMILoggedTagSet

Properties

The "LoggedTagSet" object has the following properties:

- **Count**
  Returns the number of elements of the "LoggedTagSet" list.
- **Error**
  Returns an error code for the last faulty read or write operation.

Methods

The "LoggedTagSet" object has the following methods:

- **Add()**
  Adds a logging tag to the "LoggedTagSet" list.
- **Clear()**
  Removes all logging tags from the "LoggedTagSet" list.
- **Item()**
  Returns a logging tag of the "LoggedTagSet" list.
- **Read()**
  Reads in all logging tags of the "LoggedTagSet" list.
- **Remove()**
  Removes a logging tag by its name from the "LoggedTagSet" list.

See also

TagLogging (Page 1313)

TagLogging.CreateLoggedTagSet() (Page 1314)

LoggedTag (Page 1315)

LoggedTagSet.Count

Description

The "Count" property returns the number of elements in the "LoggedTagSet" list.

Type

UInt32

Access

Read-only

Syntax

```
LoggedTagSet.Count
```

See also

LoggedTagSet (Page 1324)

LoggedTagSet.Error

Description

The "Error" property returns an error code for the last faulty read or write operation.

The value "0" is returned after a successful read or write operation. The error code always relates to the last method call of the object.

Evaluate the "Error" property for all objects of the "LoggedTagSet".

Type

ErrorCode

Access

Read-only

Syntax

```
LoggedTagSet.Error
```

See also

LoggedTagSet (Page 1324)

LoggedTagSet.Add()

Description

The "Add" method adds logging tags ("LoggedTag" objects) to the "LoggedTagSet" list. The logging tags are referenced using the name.

Syntax

```
LoggedTagSet.Add(loggedTags)
```

Parameters

`loggedTags`

Type: String, HMILoggedTag | String[], HMILoggedTag

Names of "LoggedTag" objects that are added to the list.

Note

No "LoggedTag" object can be transferred as a parameter. A "LoggedTag" object is referenced using the name.

Return value

Object[], HMILoggedTag[] (Page 1315)

See also

LoggedTag (Page 1315)

LoggedTagSet (Page 1324)

LoggedTagSet.Clear()

Description

The "Clear" method removes all logging tags ("LoggedTag" objects) from the "LoggedTagSet" list.

Syntax

```
LoggedTagSet.Clear()
```

Parameter

--

Return value

--

See also

LoggedTagSet (Page 1324)

LoggedTag (Page 1315)

LoggedTagSet.Item()

Description

The "Item()" method returns a "LoggedTag" object of the "LoggedTagSet" list.

Syntax

```
LoggedTagSet[.Item](name)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "LoggedTagSet" object.

Parameters

`name`

Type: String, HMILoggedTag | Int32, HMILoggedTag

Tag name or index number (1 ... n) of a "LoggedTag" object of the list.

Note

The index number does not describe the order in which the "LoggedTag" objects were added to the "LoggedTagSet" list.

Return

Object, HMILoggedTag (Page 1315)

See also

LoggedTagSet (Page 1324)

LoggedTag (Page 1315)

LoggedTagSet.Read()

Description

The "Read" method reads all logging tags ("LoggedTag" objects) from the "LoggedTagSet" list. The value, the Quality Code, the time stamp and context information of all logging tags are determined when the logging tag is read.

The method executes an asynchronous read operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the

successful (then()) and faulty (catch()) execution of the read operation. Depending on the result, after execution is complete, the corresponding handler of the Promise pattern is called with an array with "LoggedTagResult" object or an error code as parameter.

Syntax

```
LoggedTagSet.Read(dateFrom,dateTo,boundingValue)
.then(function(loggedTagArrayResult) {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

dateFrom

Type: DateTime

End date of the time period

dateTo

Type: DateTime

Start date of the time period

boundingValue

Type: Boolean

Specifies whether the limit values of the time period are transferred.

Return value

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  Object, HMILoggedTagResult[] (Page 1319) as parameter of the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists when all logging tags of the "LoggedTagSet" object could not be read.

See also

LoggedTagSet (Page 1324)

LoggedTag (Page 1315)

LoggedTagResult (Page 1319)

LoggedTagSet.Remove()

Description

The "Remove" method removes logging tags ("LoggedTag" objects) using their names from the "LoggedTagSet" list.

Syntax

```
LoggedTagSet.Remove(loggedTags)
```

Parameters

`loggedTags`

Type: String, HMILoggedTag | String[], HMILoggedTag

Removes a "LoggedTag" object from the "LoggedTagSet" list.

Note

No "LoggedTag" object can be transferred as a parameter. A "LoggedTag" object is referenced using the name.

Return value

--

See also

LoggedTagSet (Page 1324)

LoggedTag (Page 1315)

LoggedTag

Description

LoggedTag (Page 1315)

LoggedTagResult

Description

LoggedTagResult (Page 1319)

SysFct

Description

The "SysFct" object ("HMITagLoggingSysFct" type) enables access to system functions of the "TagLogging" object.

Object type

HMITagLoggingSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **ClearTagLog()**
  Deletes all logging tags in the specified logging system.
- **WriteManualValue()**
  Assigns a new value to the specified logging tag.

See also

TagLogging (Page 1313)

SysFct.ClearTagLog()

Description

The "ClearTagLog" method deletes the logged tags from the logging system whose name was passed via the parameter. The method removes all records from the specified logging system. All segments up to the current segment are deleted. The remaining segment is given a new start time.

Note

No automatic backup is created before the "ClearTagLog" method is executed.

The method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
HMIRuntime.TagLogging.SysFct.ClearTagLog(LogName)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
});
```

Parameters

LogName

Type: String, HMIDataLog

Name of the logging system from which the logging tags are deleted.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler

See also

SysFct (Page 1331)

SysFct.WriteManualValue()

Description

The "WriteManualValue" method assigns a new value to the specified logging tag. The associated time stamp is transferred in this process.

Syntax

```
HMIRuntime.TagLogging.SysFct.WriteManualValue(LoggedTagName,Value,Ti
mestamp);
```

Parameter

LoggedTagName

Type: String, HMILoggedTag

Logging tag which is assigned the specified value.

`Value`

Type: Variant

Value assigned to the specified logging tag.

`Timestamp`

Type: DateTime

Time stamp assigned to the specified value.

Return value

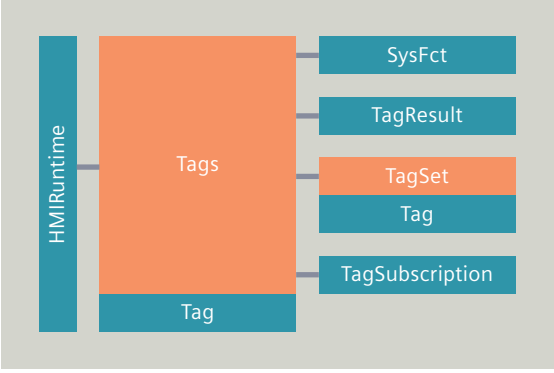
ErrorCode

See also

SysFct (Page 1331)

10.2.2.18 Tags

Description



The "Tags" object ("HMITags" type) enables access to HMI tags in runtime. By default, you reference a "Tag" object ("HMITag" type) through the "Tags" object. The "Tag" object gives you access to all properties and methods of the tags.

Use

Note

The "Tags" object is not a listing like, for example the objects "TagSet" or "AlarmSet", but rather a Factory. You create an instance of the "Tag" object via the tag name.

The "Tag" objects cannot be counted and enumerated like conventional lists.

The Tags object declares tags ("Tag" objects) for read and write access. The appropriate HMI tags must exist for the read and write access to be executed without errors.

To reduce the use of the "Tags" object, you can also use the alias `Tags` for `HMIRuntime.Tags`.

Object type

HMITags

Properties

--

Methods

The "Tags" object has the following methods:

- **CreateSubscription()**
  Creates a "TagSubscription" object for monitoring HMI tags
- **CreateTagSet()**
  Creates a new "TagSet" object.
- **Item()**
  Returns a new instance of a "Tag" object.

Tags.CreateSubscription()

Description

The "CreateSubscription" method creates a "TagSubscription" object ("HMITagSubscription" type). The "TagSubscription" returned object enables HMI tags to be monitored for change.

Syntax

```
Tags.CreateSubscription(tagNameArray, OnTag);
```

Parameters

`tagNameArray`

Type: String | String[], HMITag[]

Name or array of names of the monitored HMI tags.

`OnTag`

Type: Function, HMIOnTagCB

Callback function that is executed when HMI tags are changed.

Prototype of the callback function: `OnTag(tagResult)`

The "tagResult" parameter is an object of type "HMITagResult[]" that contains the properties of the changed HMI tags.

Return value

Object, HMITagSubscription (Page 1363)

Example

Monitor changes to HMI tags "HMI_Tag_1", "HMI_Tag_2", "HMI_Tag_3", and output names and changed values:

Copy code

```
let subs = HMIRuntime.Tags.CreateSubscription(['HMI_Tag_1', 'HMI_Tag_2',
'HMI_Tag_3'], function (TagArray) {
    for (let index in TagArray) {
        HMIRuntime.Trace("Tag Name_" + (index + 1) + " = " +
TagArray[index].Name);
        HMIRuntime.Trace("Tag Value_" + (index + 1) + "= " +
TagArray[index].Value);
    }
});
subs.Start();
```

See also

Tags (Page 1333)

TagResult (Page 1356)

TagSubscription (Page 1363)

Tags.CreateTagSet()

Description

The "CreateTagSet" method creates a new "TagSet" object ("HMITagSet" type). The "TagSet" object can be filled with one or more tags.

You use the returned "TagSet" object for read and write access to multiple tags.

Syntax

```
[HMIRuntime.]Tags.CreateTagSet([tagNameArray]);
```

Note

The `HMIRuntime.` part of the expression is not required. The alias `Tags` stands for `HMIRuntime.Tags`.

Parameters

`tagNameArray`

Optional, type: String, HMITag | String[], HMITag

Name of a tag or array with names of multiple tags that are added to the "TagSet" object. Without parameters, an empty "TagSet" object is created.

Return value

Object, HMITagSet (Page 1365)

Example

Create a "TagSet" object and add multiple objects "Tag" by their names:

Copy code

```
function TagSetCreate() {
   //Tags_Required: "Tag0"; "Tag1"; "Tag2"
   var tagnameArray = ["Tag0","Tag1","Tag2"];
   var ts = Tags.CreateTagSet(tagnameArray);
}
```

See also

Tags (Page 1333)

TagSet (Page 1365)

Tags.Item()

Description

The "Item" returns a new instance of a "Tag" object.

Syntax

```
[HMIRuntime.]Tags[.Item](tagName);
```

Note

The `HMIRuntime.` part of the expression is not required. The alias `Tags` stands for `HMIRuntime.Tags`.

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Tags" object.

Parameters

`tagName`

Type: String, HMITag

Tag name of a "Tag" object.

Note

The "Tags" object is not a list, but rather a Factory. The "Tag" objects cannot be counted and enumerated like conventional lists. Tags are only referenced through their configured name.

Return value

Object, HMITag (Page 1338)

Example

Instantiate the "TankLevel" tag as the "Tag" object and assign it to the "level" tag:

Copy code

```
var level = HMIRuntime.Tags.Item("TankLevel");
```

or

Copy code

```
var level = Tags("TankLevel");
```

See also

Tags (Page 1333)

Tag (Page 1338)

Tag

Description

The "Tag" object (type "HMITag") represents an HMI tag in runtime. A "Tag" object is returned by the "Tags" object or the "TagSet" list. The "Tag" object gives you access to all properties and methods of a tag.

Object type

HMITag

Properties

The "Tag" object has the following properties:

- **ErrorDescription**
  Returns a description of the error code for the last faulty access.
- **LastError**
  Returns an error code for the last faulty read or write operation.
- **Name**
  Returns the name of the tag.
- **QualityCode**
  Returns the quality level of a tag value after reading a tag.
- **TimeStamp**
  Returns the time stamp of the last read operation.
- **Value**
  Specifies a value for the object being used or returns it.

Methods

The "Tag" object has the following methods:

- **Decrease()**
  Reduces the current tag value in the AS by the specified value.
- **Increase()**
  Increases the current tag value in the AS by the specified value.
- **Read()**
  Reads a tag ("Tag" object).
- **ResetBit()**
  Deletes a bit of the tag in the automation system.
- **SetBit()**
  Sets a bit of the tag in the AS.
- **Write()**
  Writes tag in the automation system.

- **WriteQCD()**
  Writes the value of the tag ("Tag" object).
- **WriteWithOperatorMessage()**
  Writes the values of the tag ("Tag" object) and then triggers an operator input alarm.

Values after initialization

The properties of the "Tag" object include the following values after successful initialization of the object:

Property	After successful initialization
Name	Tag name (unchanged)
Value	VT_EMPTY
QualityCode	BAD NON-SPECIFIC
TimeStamp	0
LastError	0
ErrorDescription	""

Values after a read operation

The properties of the "Tag" object include the following values after the last read operation:

Property	After successful read operation	After unsuccessful read operation
Name	Tag name (unchanged)	
Value	Current value of the tag	VT_EMPTY
QualityCode	Quality level	BAD OUT OF SERVICE
TimeStamp	Current time stamp of the tag	0
LastError	0	Error code of read operation
ErrorDescription	""	Description of the error code

Values after a write operation

The properties of the "Tag" object include the following values after the last write operation:

Property	After successful write operation	After unsuccessful write operation
Name	Tag name (unchanged)	
Value	Current value of the "Tag" object (unchanged)	
QualityCode	BAD OUT OF SERVICE	
TimeStamp	0	
LastError	0	Error code of write operation
ErrorDescription	""	Description of the error code

Tag.ErrorDescription

Description

The "ErrorDescription" property returns a description of the error code for the last faulty access.

Note

An empty string is returned after a successful read or write operation.

Type

String

Access

Read-only

Syntax

```
Tag.ErrorDescription
```

See also

Tag (Page 1338)

Tag.LastError

Description

The "LastError" property returns an error code for the last faulty read or write operation.

Note

The value "0" is returned after a successful read or write operation.

Type

ErrorCode

Access

Read-only

Syntax

```
Tag.LastError
```

See also

Tag (Page 1338)

Tag.Name

Description

The "Name" property returns the name of the tag ("Tag" object).

Type

String, HMITag

Access

Read-only

Syntax

```
Tag.Name
```

See also

Tag (Page 1338)

Tag.QualityCode

Description

The "QualityCode" property returns the level of the quality of a tag value after the reading of a tag.

The quality code has the binary 8-bit structure **QQSSSLL.** The first two positions (QQ) of the quality code define the quality of the tag value:

Quality	Description	Q	Q	S	S	S	S	L	L
Bad	Tag value cannot be used.	0	0	-	-	-	-	-	-
Uncertain	Quality of the tag value is worse than usual. However, it might still be possible to use the tag value.	0	1	-	-	-	-	-	-
Good (Non-Cascade)	Quality of the tag value is good. Attention should be paid to substatus.	1	0	-	-	-	-	-	-
Good (Cascade)	Quality of the tag value is good. Tag value could be used.	1	1	-	-	-	-	-	-

Positions 3 to 6 (SSSS) of the quality code specify the substatus of the quality. Positions 7 and 8 (LL) are optional and define possible limits.

Note

Directly after initializing a "Tag" object, the "BAD NON-SPECIFIC" value is returned. After a write operation or an incorrect read operation, the value "BAD OUT OF SERVICE" is returned.

Type

UInt32

Access

Read-write

Syntax

```
Tag.QualityCode
```

Quality code of tags

The realized quality codes are listed in the following table. The table begins with the worst quality code and ends with the best quality code. The best quality code has the lowest priority, while the worst quality has the highest priority. If several statuses occur simultaneously for a tag in the processing chain, the poorest code is passed on.

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3F	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-
0x00	Bad	Non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain.	0	1	0	0	0	0	-	-
0xE0	Good (Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	1	0	0	0	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0xC0	Good (Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Limit

The quality codes can be further subdivided by limits. Limits are optional.

Description	Q	Q	S	S	S	S	L	L
O.K. - The value is free to move.	-	-	-	-	-	-	0	0
Low limited - The value has acceded its low limits.	-	-	-	-	-	-	0	1
High limited - The value has acceded its high limits.	-	-	-	-	-	-	1	0
Constant (high and low limited) - The value cannot move, no matter what the process does.	-	-	-	-	-	-	1	1

See also

Tag (Page 1338)

Tag.WriteQCD() (Page 1353)

Tag.TimeStamp

Description

The "TimeStamp" property returns or specifies the time stamp of the last read operation.

Note

After a write operation or an incorrect read operation, the value "0" is returned.

Type

DateTime

Access

Read-write

Syntax

```
Tag.TimeStamp
```

See also

Tag (Page 1338)

Tag.WriteQCD() (Page 1353)

Tag.Value

Description

The "Value" property specifies or returns the value of a tag.

Note

Immediately after initialization of a "Tag" object or an incorrect read operation, the value "VT_EMPTY" is returned.

Type

Variant

Access

Read-write

Syntax

```
Tag.Value
```

See also

Tag (Page 1338)

Tag.Decrease()

Description

The "Decrease" method reduces the current tag value in the automation system by the specified value. The value is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Syntax

```
Tag.Decrease(value)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

value

Type: Float

Numerical value by which the current tag value is decreased in the AS.

Return value

Promise

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Tag (Page 1338)

Tag.Increase() (Page 1347)

SysFct.DecreaseTag() (Page 1383)

Tag.Increase()

Description

The "Increase" method increases the current tag value in the AS by the specified value. The value is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read-" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Syntax

```
Tag.Increase(value)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

value

Type: Float

Numerical value by which the current tag value is increased in the AS.

Return value

Promise

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Tag (Page 1338)

Tag.Decrease() (Page 1346)

SysFct.IncreaseTag() (Page 1384)

Tag.Read()

Description

The "Read" method reads a tag ("Tag" object). The value, the Quality Code and the time stamp of the tag are determined when the tag is read.

The tags are either read from the tag image (cache) or directly from the AS. When the tag image is used, the method registers all tags that are not yet registered. You should use the tag image for cyclic readout of tags. If you do not need the tag value cyclically or the update cycle of the tag is too long, use direct readout (hmiReadDirect).

The method executes a synchronous read operation. When completed, you can use the "" properties "LastError" and "ErrorDescription" to determine if the execution was successful.

Syntax

```
Tag.Read([readType])
```

Parameters

readType

Optional, type: Int32, hmiReadType

Origin of the tag values:

- hmiReadCache (0) or empty
  Reads the tag value from the tag image. If no registration exists, the tag is registered. For high-performance access, define the used tags as triggers of the script.
- hmiReadDirect (1)
  Reads the tag value directly from the AS. The tag image is not used.

Return value

Variant

See also

Tag (Page 1338)

Tag.ResetBit()

Description

The "ResetBit" method deletes a bit of the tag in the AS. The bit of the tag is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Syntax

```
Tag.ResetBit(BitNumber)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

`BitNumber`

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of tag which is set to "0" (False).

Return value

Promise

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Tag (Page 1338)

Tag.SetBit() (Page 1351)

SysFct.ResetBitInTag() (Page 1385)

Tag.SetBit()

Description

The "SetBit" method sets a bit of the tag in the AS. The bit of the tag is written directly to the AS and not returned to the "Tag" object. If you need the changed tag value, execute a "Read" method.

The method prevents multiple transfer of the tag value for reading, calculating and writing.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, the corresponding handler of the Promise pattern with the error code as parameter is called after the execution. In case of a successful execution, the handler will always receive the error code "0".

For the method to be executed, the value of the tags must be current and valid, the quality code must correspond to Good (cascade).

The following conditions must be met for external tags:

- the connection to the PLC is set up and
- the acquisition mode of the tags is "Cyclic in operation" and
- the tag is used by an object, e.g. an I/O field.

Syntax

```
Tag.SetBit(BitNumber)
.then(function(ErrorCode) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

`BitNumber`

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of the tag that is set to "1" (TRUE).

Return value

Promise

Depending on the status of the Promise object:

- Promise successful (fulfilled)
  ErrorCode "0" as parameter of the "then()" handler
- Promise failed ("rejected")
  ErrorCode as parameter of the "catch()" handler

See also

Tag (Page 1338)

Tag.ResetBit() (Page 1350)

SysFct.SetBitInTag() (Page 1386)

Tag.Write()

Description

The "Write" method writes the values of the tag ("Tag" object). You must first set the values of the individual tags with the "Value" property. The value of the "Value" property does not have to be the current value of the tag after the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

The Tag.Write method works asynchronously. The Tag.Read method may return unexpected results if the processing of a preceding Write call is not yet complete.

If the method waits for the write operation to be completed ("hmiWriteWait"), the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method of the "TagSet" object.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Syntax

```
Tag.Write([value],[writeType])
```

Parameters

value

Optional, type: Variant

Writes the tag value:

- Specify value
  The specified value overwrites the current value of the "Value" property of the tag.
- Without value
  The current value of the "Value" property of the tag is written.

writeType

Optional, type: Int32, hmiWriteType

Specifies if the method waits for the write operation to be completed:

- hmiWriteNoWait (0) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- hmiWriteWait (1)
  Waits until the tag value is written in the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

ErrorCode

See also

Tag (Page 1338)

Tag.WriteQCD() (Page 1353)

Tag.WriteWithOperatorMessage() (Page 1354)

Tag.WriteQCD()

Description

The "WriteQCD" method writes the value of an internal tag ("Tag" object), including its quality code and time stamp, both synchronously and asynchronously.

When you call the method for an external tag, it writes the quality code and time stamp predefined by the system, not the one defined by the user.

Use this method to import already logged tag values from a runtime system to WinCC Unified.

Syntax

```
Tag.WriteQCD([value],[writeType],[TimeStamp],[QualityCode])
```

Parameters

`value`

Optional, type: Variant

Writes the tag value.

`writeType`

Optional, type: Int32, hmiWriteType

Specifies if the method waits for the write operation to be completed:

- hmiWriteNoWait (0) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- hmiWriteWait (1)
  Waits until the tag value is written in the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

`TimeStamp`

Optional, type: DateTime

Writes the time stamp.

`QualityCode`

Optional, type: UInt32

Writes the quality code.

Return value

ErrorCode

See also

Tag (Page 1338)

Tag.QualityCode (Page 1341)

Tag.TimeStamp (Page 1345)

Tag.Write() (Page 1352)

Tag.WriteWithOperatorMessage() (Page 1354)

Tag.WriteWithOperatorMessage()

Description

The "WriteWithOperatorMessage" method writes the values of the tag ("Tag" object) and then triggers an operator input alarm. The value of the "Value" property does not have to be the current value of the tags after the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

The triggered operator input alarm contains the reason for the value change, the old and new value, the user and host name and the unit.

Once the write operation is completed, the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Syntax

```
Tag.WriteWithOperatorMessage(value,reason)
```

Parameters

value

Type: Variant

Tag value. The specified value overwrites the current value of the "Value" property of the tags.

reason

Type: String

Reason for the value change of the triggered alarm.

Return value

ErrorCode

Example

Write a tag and then output the operator input alarm "Reason":

Copy code

```
var tag1 = Tags("Tag1");
var err = tag1.WriteWithOperatorMessage(5000,"Reason");
```

See also

Tag (Page 1338)

Tag.Write() (Page 1352)

Tag.WriteQCD() (Page 1353)

TagResult

Description

The "TagResult" object represents the result of a changed HMI tag.

Object type

HMITagResult

Properties

The "TagResult" object has the following properties:

- **ErrorDescription**
  Returns a description of the error code for the last faulty access.
- **LastError**
  Returns an error code for the last faulty read or write operation.
- **Name**
  Returns the name of the tag.
- **QualityCode**
  Returns the quality level of a tag value after reading a tag.
- **TimeStamp**
  Returns the time stamp of the last read operation.
- **Value**
  Specifies or returns a value, depending on the object being used.

Methods

--

TagResult.ErrorDescription

Description

The "ErrorDescription" property returns a description of the error code for the last faulty access.

Note

An empty string is returned after a successful read or write operation.

Type

String

Access

Read-only

Syntax

```
TagResult.ErrorDescription
```

See also

TagResult (Page 1356)

TagResult.LastError

Description

The "LastError" property returns an error code for the last faulty read or write operation.

Note

The value "0" is returned after a successful read or write operation.

Type

ErrorCode

Access

Read-only

Syntax

```
TagResult.LastError
```

See also

TagResult (Page 1356)

TagResult.Name

Description

The "Name" property returns the name of the tag ("Tag" object).

Type

String, HMITag

Access

Read-only

Syntax

```
TagResult.Name
```

See also

TagResult (Page 1356)

TagResult.QualityCode

Description

The "QualityCode" property returns the quality level of a tag value after reading a tag.

The quality code has the binary 8-bit structure **QQSSSLL.** The first two positions (QQ) of the quality code define the quality of the tag value:

Quality	Description	Q	Q	S	S	S	S	L	L
Bad	Tag value cannot be used.	0	0	-	-	-	-	-	-
Uncertain	Quality of the tag value is worse than usual. However, it might still be possible to use the tag value.	0	1	-	-	-	-	-	-
Good (Non-Cascade)	Quality of the tag value is good. Attention should be paid to substatus.	1	0	-	-	-	-	-	-
Good (Cascade)	Quality of the tag value is good. Tag value could be used.	1	1	-	-	-	-	-	-

Positions 3 to 6 (SSSS) of the quality code specify the substatus of the quality. Positions 7 and 8 (LL) are optional and define possible limits.

Note

Directly after initializing a "Tag" object, the "BAD NON-SPECIFIC" value is returned. After a write operation or an incorrect read operation, the value "BAD OUT OF SERVICE" is returned.

Type

UInt32

Access

Read-write

Syntax

```
TagResult.QualityCode
```

Quality code of tags

The implemented quality codes are listed in the following table. The table begins with the worst quality code and ends with the best quality code. The best quality code has the lowest priority, while the worst quality has the highest priority. If multiple statuses occur simultaneously for a tag in the processing chain, the poorest code is passed on.

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x23	Bad	Device passivated - Diagnostic alerts inhibited	0	0	1	0	0	0	1	1
0x3F	Bad	Function check - Local override	0	0	1	1	1	1	1	1
0x1C	Bad	Out of Service - The value is not reliable because the block is not being evaluated, and may be under construction by a configurer. Set if the block mode is O/S.	0	0	0	1	1	1	-	-
0x73	Uncertain	Simulated value - Start	0	1	1	1	0	0	1	1
0x74	Uncertain	Simulated value - End	0	1	1	1	0	1	-	-
0x84	Good (Non-Cascade)	Active Update event - Set if the value is good and the block has an active Update event.	1	0	0	0	0	1	-	-
0x24	Bad	Maintenance alarm - More diagnostics available.	0	0	1	0	0	1	-	-
0x18	Bad	No Communication, with no usable value - Set if there has never been any communication with this value since it was last "Out of Service".	0	0	0	1	1	0	-	-
0x14	Bad	No Communication, with last usable value - Set if this value had been set by communication, which has now failed.	0	0	0	1	0	1	-	-
0x0C	Bad	Device Failure - Set if the source of the value is affected by a device failure.	0	0	0	0	1	1	-	-
0x10	Bad	Sensor failure	0	0	0	1	0	0	-	-
0x08	Bad	Not Connected - Set if this input is required to be connected and is not connected.	0	0	0	0	1	0	-	-
0x04	Bad	Configuration Error - Set if the value is not useful because there is some inconsistency regarding the parameterization or configuration, depending on what a specific manufacturer can detect.	0	0	0	0	0	1	-	-
0x00	Bad	Non-specific - There is no specific reason why the value is bad. Used for propagation.	0	0	0	0	0	0	-	-
0x28	Bad	Process related - Substitute value	0	0	1	0	1	0	-	-
0x2B	Bad	Process related - No maintenance	0	0	1	0	1	0	1	1
0x68	Uncertain	Maintenance demanded	0	1	1	0	1	0	-	-
0x60	Uncertain	Simulated value - Set when the process value is written by the operator while the block is in manual mode.	0	1	1	0	0	0	-	-
0x64	Uncertain	Sensor calibration	0	1	1	0	0	1	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x5C	Uncertain	Configuration error	0	1	0	1	1	1	-	-
0x58	Uncertain	Sub-normal	0	1	0	1	1	0	-	-
0x54	Uncertain	Engineering Unit Range Violation - Set if the value lies outside of the set of values defined for this parameter. The Limits define which direction has been exceeded.	0	1	0	1	0	1	-	-
0x50	Uncertain	Sensor conversion not accurate	0	1	0	1	0	0	-	-
0x4B	Uncertain	Substitute (constant)	0	1	0	0	1	0	1	1
0x78	Uncertain	Process related - No maintenance	0	1	1	1	1	0	-	-
0x4C	Uncertain	Initial Value - Value of volatile parameters during and after reset of the device or of a parameter.	0	1	0	0	1	1	-	-
0x48	Uncertain	Substitute value - Predefined value is used instead of the calculated one. This is used for fail safe handling.	0	1	0	0	1	0	-	-
0x44	Uncertain	Last Usable Value - Whatever was writing this value has stopped doing so. This is used for fail safe handling.	0	1	0	0	0	1	-	-
0x40	Uncertain	Non-specific - There is no specific reason why the value is uncertain.	0	1	0	0	0	0	-	-
0xE0	Good (Cascade)	Initiate Fail Safe (IFS) - The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe.	1	1	1	0	0	0	-	-
0xD8	Good (Cascade)	Local Override (LO) - The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control must be propagated to a function running in a host system for alarm and display purposes. This also implies "Not Invited".	1	1	0	1	1	0	-	-
0xD4	Good (Cascade)	Do Not Select (DNS) - The value is from a block which should not be selected, due to conditions in or above the block.	1	1	0	1	0	1	-	-
0xCC	Good (Cascade)	Not Invited (NI) - The value is from a block which does not have a target mode that would use this input.	1	1	0	0	1	1	-	-
0xC8	Good (Cascade)	Initialization Request (IR) - The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.	1	1	0	0	1	0	-	-
0xC4	Good (Cascade)	Initialization Acknowledge (IA) - The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).	1	1	0	0	0	1	-	-
0xC0	Good (Cascade)	OK - No error or special condition is associated with this value.	1	1	0	0	0	0	-	-
0xA0	Good (Non-Cascade)	Initiate fail safe	1	0	1	0	0	0	-	-

Code (hex)	Quality	Description	Q	Q	S	S	S	S	L	L
0x98	Good (Non-Cascade)	Unacknowledged Critical Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.	1	0	0	1	1	0	-	-
0x94	Good (Non-Cascade)	Unacknowledged Advisory Alarm - Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.	1	0	0	1	0	1	-	-
0x90	Good (Non-Cascade)	Unacknowledged Update event - Set if the value is good and the block has an unacknowledged Update event.	1	0	0	1	0	0	-	-
0x8C	Good (Non-Cascade)	Active Critical Alarm - Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.	1	0	0	0	1	1	-	-
0x88	Good (Non-Cascade)	Active Advisory Alarm - Set if the value is good and the block has an active Alarm with a priority less than 8.	1	0	0	0	1	0	-	-
0xA8	Good (Non-Cascade)	Maintenance demanded	1	0	1	0	1	0	-	-
0xA4	Good (Non-Cascade)	Maintenance required	1	0	1	0	0	1	-	-
0xBC	Good (Non-Cascade)	Function check - Local override	1	0	1	1	1	1	-	-
0x80	Good (Non-Cascade)	OK - No error or special condition is associated with this value.	1	0	0	0	0	0	-	-

Limit

The quality codes can be further subdivided by limits. Limits are optional.

Description	Q	Q	S	S	S	S	L	L
O.K. - The value is free to move.	-	-	-	-	-	-	0	0
Low limited - The value has acceded its low limits.	-	-	-	-	-	-	0	1
High limited - The value has acceded its high limits.	-	-	-	-	-	-	1	0
Constant (high and low limited) - The value cannot move, no matter what the process does.	-	-	-	-	-	-	1	1

See also

TagResult (Page 1356)

TagResult.TimeStamp

Description

The "TimeStamp" property returns or specifies the time stamp of the last read operation.

Note

After a write operation or an incorrect read operation, the value "0" is returned.

Type

DateTime

Access

Read-write

Syntax

```
TagResult.TimeStamp
```

See also

TagResult (Page 1356)

TagResult.Value

Description

The "Value" property specifies or returns the value of a tag.

Note

Immediately after initialization of a "Tag" object or an incorrect read operation, the value "VT_EMPTY" is returned.

Type

Variant

Access

Read-write

Syntax

```
TagResult.Value
```

See also

TagResult (Page 1356)

TagSubscription

Description

The "TagSubscription" object enables HMI tags to be monitored for change.

Use

The "AlarmSubscription" object represents a selection of active alarms. A "TagSubscription" object is initialized through the "CreateSubscription" method of the "Tags" object.

Object type

HMITagSubscription

Properties

--

Methods

The "TagSubscription" object has the following methods:

- **Start()**
  Activates monitoring of HMI tags of the "TagSubscription" object.
- **Stop()**
  Stops monitoring of HMI tags of the "TagSubscription" object.

TagSubscription.Start()

Description

The "Start" method activates the monitoring of HMI tags of the "TagSubscription" object.

Syntax

```
TagSubscription.Start();
```

Parameters

--

Return value

ErrorCode

Example

Monitor changes to the "HMI_Tag_1" and "HMI_Tag_1" tags:

Copy code

```
const subs = HMIRuntime.Tags.CreateSubscription(['HMI_Tag_1', 'HMI_Tag_2'], (changedTags)
=> {
   for (const tag of changedTags) {
          HMIRuntime.Trace(`Tag ${tag.Name} value updated to ${tag.Value}`);
   }
   subs.Stop();
});
subs.Start();
```

See also

TagSubscription (Page 1363)

TagSubscription.Stop() (Page 1364)

TagSubscription.Stop()

Description

The "Stop" method terminates the monitoring of HMI tags of the "TagSubscription" object.

Syntax

```
TagSubscription.Stop();
```

Parameters

--

Return value

ErrorCode

Example

Stop monitoring the changes to the "HMI_Tag_1" and "HMI_Tag_1" tags:

Copy code

```
const subs = HMIRuntime.Tags.CreateSubscription(['HMI_Tag_1', 'HMI_Tag_2'], (changedTags)
=>
   {
   for(const tag of changedTags)
      {
      HMIRuntime.Trace(`Tag ${tag.Name} value updated to ${tag.Value}`);
      }
   subs.Stop();
   });
subs.Start();
```

See also

TagSubscription (Page 1363)

TagSubscription.Start() (Page 1363)

TagSet

Description

The "TagSet" object ("HMITagSet" type) is a list of "Tag" objects that provides optimized access to tags in runtime. After initialization of the "TagSet" object, you can execute read and write access to multiple tags in one call. Access demonstrates better performance and lower communication load than single access to multiple tags.

You reference a "TagSet" object through the "Tags" object or create a new "TagSet" object with the "Tags.CreateTagSet" method.

By default, you access a "Tag" object (type "HMITag") through the "TagSet" object. The "Tag" object gives you access to all properties and methods of the tags.

Use

The "TagSet" object is a list and can be counted and enumerated. You can access the "TagSet" list using the index or the tag name.

The appropriate HMI tags must exist for the read and write access to tags ("Tag" objects) of the list to be executed without errors. If a read or write access error has occurred, can be read out with the properties "LastError" and "ErrorDescription" once the methods have been executed.

Object type

HMITagSet

Properties

The "TagSet" object has the following properties:

- **Count**
  Returns the number of elements of the "TagSet" list.
- **ErrorDescription**
  Returns a description of the error code for the last faulty access.
- **LastError**
  Returns an error code for the last faulty read or write operation.

Methods

The "TagSet" object has the following methods:

- **Add()**
  Adds a tag to the "TagSet" list.
- **Clear()**
  Removes all tags from the "TagSet" list.
- **Item()**
  Returns a tag of the "TagSet" list.
- **Read()**
  Reads in all tags of the "TagSet" list.
- **ReadAsync()**
  Reads in all tags of the "TagSet" list (asynchronous).
- **ReadMaxAge()**
  Reads in all tags of the "TagSet" list and ensures that these are not older than the specified time period (maxAge).
- **Remove()**
  Removes a tag by its name from the "TagSet" list.
- **Write()**
  Writes the values of all tags of the "TagSet" list.
- **WriteAsync()**
  Writes the values of all tags of the "TagSet" list (asynchronous).
- **WriteAsyncQCD()**
  Writes the values of all tags in the "TagSet" list, including their quality code and time stamps (asynchronous).
- **WriteQCD()**
  Writes the values of all tags in the "TagSet" list, including their quality code and time stamps.
- **WriteWithOperatorMessage()**
  Writes the values of all tags of the "TagSet" list and then triggers an operator input alarm for each tag.

TagSet.Count

Description

The "Count" property returns the number of elements in the "TagSet" list.

Type

UInt32

Access

Read-only

Syntax

```
TagSet.Count
```

See also

TagSet (Page 1365)

TagSet.ErrorDescription

Description

The "ErrorDescription" property returns a description of the error code for the last faulty access. The description of the error code of the last method call is returned.

Note

An empty string is returned after a successful read or write operation.

Type

String

Access

Read-only

Syntax

```
TagSet.ErrorDescription
```

See also

TagSet (Page 1365)

TagSet.LastError

Description

The "LastError" property returns an error code for the last faulty read or write operation. The error code for the last method call is returned.

Note

The value "0" is returned after a successful read or write operation.

Type

ErrorCode

Access

Read-only

Syntax

```
TagSet.LastError
```

Note

Evaluate the "LastError" property for all objects of the TagSet.

See also

TagSet (Page 1365)

TagSet.Add()

Description

The "Add" method adds a tag ("Tag" object) to the "TagSet" list. The tags are referenced by the name.

Syntax

```
TagSet.Add(tag)
```

Parameters

tag

Type: String, HMITag | String[], HMITag | Variant[][], HMITag

Names of "Tag" objects that are added to the list.

Note

No "Tag" object can be transferred as a parameter. A "Tag" object is referenced using the name.

Return value

Object[], HMITag (Page 1338)

See also

Tag (Page 1338)

TagSet (Page 1365)

TagSet.Clear()

Description

The "Clear" method removes all tags ("Tag" objects) from the "TagSet" list.

Syntax

```
TagSet.Clear()
```

Parameter

--

Return value

--

See also

TagSet (Page 1365)

TagSet.Remove() (Page 1374)

TagSet.Item()

Description

The "Item" method returns a "Tag" object of the "TagSet" list.

Syntax

```
TagSet[.Item](name)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "TagSet" object.

Parameter

`name`

Type: String, HMITag | Int32, HMITag

Tag name or index number (1...n) of a "Tag" object in the list

Note

The index number of a "Tag" object does not represent the order in which the "Tag" objects were added to the TagSet list.

Return value

Object, HMITag (Page 1338)

See also

Tag (Page 1338)

TagSet (Page 1365)

TagSet.Read()

Description

The "Read" method reads in all tags ("Tag" objects) of the "TagSet" list. The value, the Quality Code and the time stamp of all tags are determined when the tag is read.

The tags are either read from the tag image (cache) or directly from the AS. When the tag image is used, the method registers all tags that are not yet registered. You should use the

tag image for cyclic readout of tags. If you do not need the tag value cyclically or the update cycle of the tag is too long, use direct readout (hmiReadDirect).

The method executes a synchronous read operation. When completed, you can use the "TagSet" properties "LastError" and "ErrorDescription" to determine if the execution was successful.

Syntax

```
TagSet.Read([readType])
```

Parameter

readType

Optional, type: Int32, hmiReadType

Origin of the tag values:

- hmiReadCache (0) or empty
  Reads the tag value from the tag image. If no registration exists, the tag is registered. For high-performance access, define the used tags as triggers of the script.
- hmiReadDirect (1)
  Reads the tag value directly from the AS. The tag image is not used.

Return value

--

See also

TagSet (Page 1365)

TagSet.ReadAsync() (Page 1371)

TagSet.ReadMaxAge() (Page 1373)

TagSet.ReadAsync()

Description

The "ReadAsync" method reads in all tags ("Tag" objects) of the "TagSet" list. The value, the Quality Code and the time stamp of all tags are determined when the tag is read.

The tags are either read from the tag image (cache) or directly from the AS. When the tag image is used, the method registers all tags that are not yet registered. You should use the tag image for cyclic readout of tags. If you do not need the tag value cyclically or the update cycle of the tag is too long, use direct readout (hmiReadDirect).

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result,

once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter.

Syntax

```
TagSet.ReadAsync([readType])
.then(function(HMITagSet) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

readType

Optional, type: Int32, hmiReadType

Origin of the tag values:

- hmiReadCache (0) or empty
  Reads the tag value from the tag image. If no registration exists, the tag is registered. For high-performance access, define the used tags as triggers of the script.
- hmiReadDirect (1)
  Reads the tag value directly from the AS. The tag image is not used.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled
  Object, HMITagSet (Page 1365) as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists when all tags of the "TagSet" object could not be read.

See also

TagSet (Page 1365)

TagSet.Read() (Page 1370)

TagSet.ReadMaxAge() (Page 1373)

TagSet.ReadMaxAge()

Description

The "ReadMaxAge" method reads in all tags ("Tag" objects) of the "TagSet" list and ensures that these are not older than the specified time period (maxAge). The value, the Quality Code and the time stamp of all tags are determined when the tag is read.

The tags are read either from the process image (maxAge > 0) or directly from the AS (maxAge = 0). The method does not use the tag image and does not register tags. You should not use this method for cyclic readout of tags.

The method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter.

Syntax

```
TagSet.ReadMaxAge(maxAge)
.then(function(HMITagSet) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

maxAge

Type: UInt32

Time interval in milliseconds after which a tag value must be updated.

- maxAge = 0
  Read tag value immediately directly from the AS.
- maxAge > 0
  Read tag value from process image according to time stamp.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled
  Object, HMITagSet (Page 1365) as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists when all tags of the "TagSet" object could not be read.

See also

TagSet (Page 1365)

TagSet.Read() (Page 1370)

TagSet.ReadAsync() (Page 1371)

TagSet.Remove()

Description

The "Remove" method removes tags ("Tag" objects) using their names from the "TagSet" list.

Syntax

```
TagSet.Remove(tag)
```

Parameters

tag

Type: String, HMITag | String[], HMITag | Variant[][], HMITag

Removes a "Tag" object from the "TagSet" list.

Note

No "Tag" object can be transferred as a parameter. A "Tag" object is referenced using the name.

Return value

--

See also

TagSet (Page 1365)

TagSet.Clear() (Page 1369)

TagSet.Write()

Description

The "Write" method writes the values of all tags ("Tag" objects) of the "TagSet" list. You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed ("hmiWriteWait"), the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Syntax

```
TagSet.Write([writeType])
```

Parameters

writeType

Optional, type: Int32, hmiWriteType

Specifies if the method waits for the write operation to be completed:

- hmiWriteNoWait (0) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- hmiWriteWait (1)
  Waits until the tag value is written in the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

--

See also

TagSet (Page 1365)

TagSet.WriteAsync() (Page 1375)

TagSet.WriteAsyncQCD() (Page 1377)

TagSet.WriteQCD() (Page 1378)

TagSet.WriteWithOperatorMessage() (Page 1379)

TagSet.WriteAsync()

Description

The "WriteAsync" method writes the values of all tags ("Tag" objects) of the "TagSet" list. You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed ("hmiWriteWait"), the properties "LastError" and "ErrorDescription" are written for each tag. This enables you to determine if the execution was successful.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter. An execution is only faulty ("Promise rejected") when none of the tags of the "TagSet" object could be written.

Syntax

```
TagSet.WriteAsync([writeType])
.then(function(HMITagSet) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

`writeType`

Optional, type: Int32, hmiWriteType

Specifies if the method waits for the write operation to be completed:

- hmiWriteNoWait (0) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- hmiWriteWait (1)
  Waits until the tag value is written in the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled
  Object, HMITagSet (Page 1365) as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists when none of the tags of the "TagSet" object could be written.

See also

TagSet (Page 1365)

TagSet.Write() (Page 1374)

TagSet.WriteAsyncQCD() (Page 1377)

TagSet.WriteQCD() (Page 1378)

TagSet.WriteWithOperatorMessage() (Page 1379)

TagSet.WriteAsyncQCD()

Description

The "WriteAsync" method writes the values of all tags ("Tag" objects) of the "TagSet" list, including their quality code and time stamps.

You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed ("hmiWriteWait"), the properties "LastError" and "ErrorDescription" are written for each tag. This enables you to determine if the execution was successful.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

The method executes an asynchronous write operation without blocking further script execution. In doing so, the method uses a Promise object, which has handlers for successful ("then()") and faulty ("catch()") execution of the write operation. Depending on the result, once execution is complete the corresponding handler of the Promise pattern is called with the "TagSet" object or the error code as parameter. An execution is only faulty ("Promise rejected") when none of the tags of the "TagSet" object could be written.

When you call the method for external tags, it writes the quality code and time stamp predefined by the system, not the one defined by the user.

Syntax

```
TagSet.WriteAsyncQCD([writeType])
.then(function(HMITagSet) {
    ...
})
.catch(function(ErrorCode) {
    ...
})
```

Parameters

writeType

Optional, type: Int32, hmiWriteType

Specifies if the method waits for the write operation to be completed:

- hmiWriteNoWait (0) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- hmiWriteWait (1)
  Waits until the tag value is written in the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

Promise

Depending on the status of the Promise object:

- Promise fulfilled
  Object, HMITagSet (Page 1365) as parameter of the "then()" handler.
- Promise failed (rejected)
  ErrorCode as parameter of the "catch()" handler. This status only exists when none of the tags of the "TagSet" object could be written.

See also

TagSet (Page 1365)

TagSet.Write() (Page 1374)

TagSet.WriteAsync() (Page 1375)

TagSet.WriteQCD() (Page 1378)

TagSet.WriteWithOperatorMessage() (Page 1379)

TagSet.WriteQCD()

Description

The "Write" method writes the values of all tags ("Tag" objects) of the "TagSet" list, including their quality code and time stamps.

You must first set the values of the individual tags with the "Value" property. The value of the "Value" property must not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

If the method waits for the write operation to be completed ("hmiWriteWait"), the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

When you call the method for external tags, it writes the quality code and time stamp predefined by the system, not the one defined by the user.

Syntax

```
TagSet.WriteQCD([writeType])
```

Parameter value

writeType

Optional, type: Int32, hmiWriteType

Specifies if the method waits for the write operation to be completed:

- hmiWriteNoWait (0) or empty
  Writes the tag value without waiting. Errors for the write operation are not detected.
- hmiWriteWait (1)
  Waits until the tag value is written in the AS. The properties "LastError" and "ErrorDescription" of the tags are written.

Return value

--

See also

TagSet (Page 1365)

TagSet.Write() (Page 1374)

TagSet.WriteAsync() (Page 1375)

TagSet.WriteAsyncQCD() (Page 1377)

TagSet.WriteWithOperatorMessage() (Page 1379)

TagSet.WriteWithOperatorMessage()

Description

The "WriteWithOperatorMessage" method writes the values of all tags ("Tag" objects) of the "TagSet" list and then triggers an operator input alarm for each tag. You must first set the values of the individual tags with the "Value" property. The value of the "Value" property does not correspond to the actual value of the tag once the write operation is complete. If you want to update the information for the tags, execute a "Read" method.

In addition to the reason, the triggered operator input alarms contain the old and new value, the user and host name and the unit.

Once the write operation is completed, the properties "LastError" and "ErrorDescription" are also written for each tag. This enables you to determine if the execution was successful. If you need the result of the write operation without blocking the script execution, use the "WriteAsync" method.

The properties "QualityCode" and "TimeStamp" of the tags are not determined during writing.

The tags are written directly to the AS. The tag image and the process image are not used by the method.

Syntax

```
TagSet.WriteWithOperatorMessage(reason)
```

Parameters

reason

Type: String

Reason for the change in value of the triggered alarms.

Return value

ErrorCode

Example

Write tags in a TagSet and then output the operator input alarm "Reason":

Copy code

```
var ts = Tags.CreateTagSet();
ts.Add("Tag1"); ts.Add("Tag2");
ts.Item("Tag1").Value = 10; ts.Item("Tag2").Value = 20;
var err = tag1.WriteWithOperatorMessage("Reason");
```

See also

TagSet (Page 1365)

TagSet.Write() (Page 1374)

TagSet.WriteAsync() (Page 1375)

TagSet.WriteAsyncQCD() (Page 1377)

TagSet.WriteQCD() (Page 1378)

Tag

Description

Tag (Page 1338)

SysFct

Description

The "SysFct" object ("HMITagSysFct" type) enables access to the system functions of the "Tags" object.

Object type

HMITagSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **DecreaseTag()**
  Subtracts a value from the value of the tag.
- **IncreaseTag()**
  Adds a value to the value of the tag.
- **InvertBitInTag()**
  Inverts a bit in the tag.
- **ResetBitInTag()**
  Resets a bit in the tag.
- **SetBitInTag()**
  Sets a bit in the tag.
- **SetTagValue()**
  Assigns a new value to the tag.
- **ShiftAndMask()**
  Converts the bit pattern of a source tag into the bit pattern of a target tag.
- **UpdateTag()**
  Reads from the PLC the current value of the tags with the specified update ID.

See also

Tags (Page 1333)

SysFct.CreateSetTagCommand()

Description

The "CreateSetTagCommand" method is an auxiliary function via which the result of return parameters of system functions can be written to HMI tags.

Syntax

```
[HMIRuntime.]Tags.SysFct.CreateSetTagCommand(TagName);
```

Parameters

TagName

Type: String, HMITag

HMI tag to which the return is written.

Return value

Object, HMISetTagCommand

Example

Write the result of the "LookUpText" system function to the "tempResultTagName" HMI tag and read it out:

Copy code

```
let setTagCommand = HMIRuntime.Tags.SysFct.CreateSetTagCommand(tempResultTagName);
HMIRuntime.Resources.SysFct.LookUpText(setTagCommand, qualitiyValue.value,
HMIRuntime.Language, nameTextList);
result = Tags(tempResultTagName).Read();
```

See also

SysFct (Page 1381)

SysFct.DecreaseTag()

Description

The "DecreaseTag" method subtracts the given value from the value of the tag.

Note

The system function uses the same tag as input and output values. When this system function is used to convert a value, an auxiliary tag must be used. You can assign the tag value to the auxiliary tag with the "SetTagValue" system function.

Syntax

```
[HMIRuntime.]Tags.SysFct.DecreaseTag(Tag,value);
```

Parameters

`Tag`

Type: String, HMITag

Tag from which the specified value is subtracted.

`value`

Type: Float

Numeric value that is subtracted.

Return value

ErrorCode

See also

SysFct (Page 1381)

SysFct.IncreaseTag() (Page 1384)

Tag.Decrease() (Page 1346)

SysFct.IncreaseTag()

Description

The "IncreaseTag" method adds the specified value to the value of the tag.

Note

The system function uses the same tag as input and output values. When this system function is used to convert a value, an auxiliary tag must be used. You can assign the tag value to the auxiliary tag with the "SetTagValue" system function.

Syntax

```
[HMIRuntime.]Tags.SysFct.IncreaseTag(Tag,value);
```

Parameters

`Tag`

Type: String, HMITag

Tag to which the specified value is added.

`value`

Type: Float

Numeric value that is added.

Return value

ErrorCode

See also

SysFct (Page 1381)

SysFct.DecreaseTag() (Page 1383)

Tag.Increase() (Page 1347)

SysFct.InvertBitInTag()

Description

The "InvertBitInTag" method inverts a bit in the specified tag:

- If the bit in the tag has the value "1" (True), it is set to "0" (False).
- If the bit in the tag has the value "0" (False), it is set to "1" (True).

After changing the specified bit, the system function transfers the entire tag back to the PLC. In the meantime it is not checked whether other bits in the tag have changed. Operator and PLC have read-only access to the specified tag until it is transferred back to the PLC.

Syntax

```
[HMIRuntime.]Tags.SysFct.InvertBitInTag(Tag,BitNumber);
```

Parameters

Tag

Type: String, HMITag

tag in which the bit is inverted.

BitNumber

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of the tag that is inverted.

The bits are counted in a tag from right to left. The counting begins with 0.

Return value

ErrorCode

See also

SysFct (Page 1381)

SysFct.ResetBitInTag() (Page 1385)

SysFct.SetBitInTag() (Page 1386)

SysFct.ResetBitInTag()

Description

The "ResetBitInTag" method sets a bit in the specified tag to "0" (False).

After changing the specified bit, the system function transfers the entire tag back to the PLC. In the meantime it is not checked whether other bits in the tag have changed. Operator and PLC have read-only access to the specified tag until it is transferred back to the PLC.

Syntax

```
[HMIRuntime.]Tags.SysFct.ResetBitInTag(Tag,BitNumber);
```

Parameters

`Tag`

Type: String, HMITag

Tag in which the bit is set.

`BitNumber`

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of tag which is set to "0" (False).

The bits are counted in a tag from right to left. The counting begins with 0.

Return value

ErrorCode

See also

SysFct (Page 1381)

SysFct.InvertBitInTag() (Page 1384)

SysFct.SetBitInTag() (Page 1386)

Tag.ResetBit() (Page 1350)

SysFct.SetBitInTag()

Description

The "SetBitInTag" method sets a bit in the specified tag to "1" (True).

After changing the specified bit, the system function transfers the entire tag back to the PLC. In the meantime it is not checked whether other bits in the tag have changed. Operator and PLC have read-only access to the specified tag until it is transferred back to the PLC.

Note

This function does not support data types of type UInt.

Syntax

```
[HMIRuntime.]Tags.SysFct.SetBitInTag(Tag,BitNumber);
```

Parameters

`Tag`

Type: String, HMITag

Tag in which the bit is set.

BitNumber

Type: UInt8, value range: 0-63, depending on the data type of the tag

Bit of tag which is set to "1" (True).

The bits are counted in a tag from right to left. The counting begins with 0.

Return value

ErrorCode

See also

SysFct (Page 1381)

SysFct.InvertBitInTag() (Page 1384)

SysFct.ResetBitInTag() (Page 1385)

Tag.SetBit() (Page 1351)

SysFct.SetTagValue()

Description

The "SetTagValue" method assigns a new value to the specified tag.

Note

This system function can be used to assign strings and numbers, depending on the type of tag.

Note

The system function is only executed after a connection has been established.

Syntax

```
[HMIRuntime.]Tags.SysFct.SetTagValue(Tag,Value);
```

Parameters

Tag

Type: String, HMITag

Tag to which a value is assigned.

Value

Type: Variant

Value that is assigned.

Return value

ErrorCode

See also

SysFct (Page 1381)

SysFct.ShiftAndMask()

Description

The "ShiftAndMask" method converts the input bit pattern of a source tag into the output bit pattern of a target tag. This involves bit shifting and masking.

Note

If the source and target tag have a different number of bits, using the system function in the target tag can result in a violation of the value range.

Syntax

```
[HMIRuntime.]Tags.SysFct.ShiftAndMask(Source,Target,bitsToShift,bitP
attern);
```

Parameter

`Source`

Type: String, HMITag

Source tag that receives the input bit pattern. Tags of integer type are allowed.

The source and target tags must have the same number of bits.

`Target`

Type: String, HMITag

Target tag to which the output bit pattern is saved. Tags of integer type are allowed.

The source and target tags must have the same number of bits.

`bitsToShift`

Type: Int8

Number of bits by which the input bit pattern is shifted right. A negative value shifts the input bit pattern to the left.

The number of bits to be shifted must be less than the number of bits of the source and target tags.

Note

The left bit is "1" in a source tag of the data type with negative signed integer. This sign bit is padded with "0" when the bits are shifted right. The sign changes to "+".

bitPattern

Type: UInt32

Integer number that is used as bit mask. The bit pattern is used to multiply the shifted input bit pattern.

The bit mask must not have more bits than the source and target tags.

Return value

ErrorCode

Example

The source tag of the 16-bit integer type has the current value "72": 0000000001001000.

Shift bits

`bitsToShift` has the value "+3". The input bit pattern is shifted 3 bits to the right. Left is filled with "0". Three bits are truncated on the right: 0000000000001001. The new decimal value is "9".

Mask bits

`bitPattern` has the value "2478" with the bit pattern "0000100110101110". The shifted input bit pattern (0000000000001001) is multiplied by the bit mask, with bit-by-bit logical AND operation. The result is the decimal value "8".

Save result

The decimal value is stored in the target tag.

See also

SysFct (Page 1381)

SysFct.UpdateTag()

Description

The "UpdateTag" method reads from the PLC the current value of the tags with the specified update ID. An update ID can be used for several tags.

Note

Tags of the data type STRUCT are not supported.

Syntax

```
[HMIRuntime.]Tags.SysFct.UpdateTag(UpdateID);
```

Parameter

`UpdateID`

Type: UInt32, HMITagUpdateID

Specifies the Update ID.

Return value

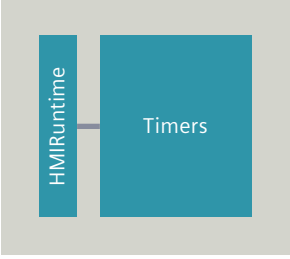
ErrorCode

See also

SysFct (Page 1381)

10.2.2.19 Timers

Description



The "Timers" object can time the script run by means of one-time or cyclic timers. You can execute individual functions delayed or repeatedly.

Object type

HMITimers

Properties

--

Methods

The "Timers" object has the following methods:

- **ClearInterval()**
  Deletes a timer object for cyclic execution of a function.
- **ClearTimeout()**
  Deletes a timer object for delayed execution of a function.
- **SetInterval()**
  Creates a timer object for the cyclic execution of a function.
- **SetTimeout()**
  Creates a timer object for delayed execution of a function.

Example

Execution of the "myFunc" function is delayed:

Copy code

```
function setDelay() {
    var timerId = HMIRuntime.Timers.SetTimeout(myFunc, 5000);
    function myFunc() {
        HMIRuntime.Trace("SetTimeout triggered");
    }
}
```

Timers.ClearInterval()

Description

The "ClearIntervall" method deletes a timer object for cyclic execution of a function. You can delete a timer object at any time prior to the next execution.

Syntax

```
HMIRuntime.Timers.ClearIntervall(TimerID);
```

Parameters

TimerID

Type: Int32

ID of the timer object to be deleted. The ID is returned by the "SetInterval" method during creation of the timer object.

Return value

--

Example

Create and delete a cyclic timer:

Copy code

```
var TimerID = HMIRuntime.Timers.SetInterval(update, 1000);
HMIRuntime.Timers.ClearInterval(TimerID);
```

See also

Timers (Page 1390)

Timers.SetInterval() (Page 1393)

Timers.ClearTimeout()

Description

The "ClearTimeout" method deletes a timer object for delayed execution of a function. You can delete a timer object at any time prior to the execution.

Syntax

```
HMIRuntime.Timers.ClearTimeout(TimerID);
```

Parameters

TimerID

Type: Int32

ID of the timer object to be deleted. The ID is returned by the "SetTimeout" method during creation of the timer object.

Return value

--

Example

Create and delete a one-time timer:

Copy code

```
var TimerID = HMIRuntime.Timers.SetTimeout(update,1000);
HMIRuntime.Timers.ClearTimeout(TimerID);
```

See also

Timers (Page 1390)

Timers.SetTimeout() (Page 1394)

Timers.SetInterval()

Description

The "SetIntervall" method creates a timer object for cyclic execution of a function and returns the ID of the timer object. When a time interval expires, a function is started and scheduled for a new execution according to the time interval.

Syntax

```
HMIRuntime.Timers.SetIntervall(CallbackFunction, DelayInMillisecs);
```

Parameters

CallbackFunction

Type: Function, HMIOnTimerCB

Function that is executed cyclically.

Required prototype of the callback function: `TimerCallback()`

DelayInMillisecs

Type: UInt32

Time interval in milliseconds in which the function is executed cyclically.

Return value

Int32

Example

Create a cyclic timer:

Copy code

```
var TimerID = HMIRuntime.Timers.SetInterval(update,1000);
```

See also

Timers (Page 1390)

Timers.ClearInterval() (Page 1391)

Timers.SetTimeout()

Description

The "SetTimeout" method creates a timer object for time-delayed execution of a function and returns the ID of the timer object. When a time interval has elapsed, a function is started.

Syntax

```
HMIRuntime.Timers.SetTimeout(CallbackFunction, DelayInMillisecs);
```

Parameters

CallbackFunction

Type: Function, HMIOnTimerCB

Function that is executed once.

Required prototype of the callback function: `TimerCallback()`

DelayInMillisecs

Type: UInt32

Time interval in milliseconds after which the function is executed delayed.

Return value

Int32

Example

Create a one-time timer:

Copy code

```
var TimerID = HMIRuntime.Timers.SetInterval(update,1000);
```

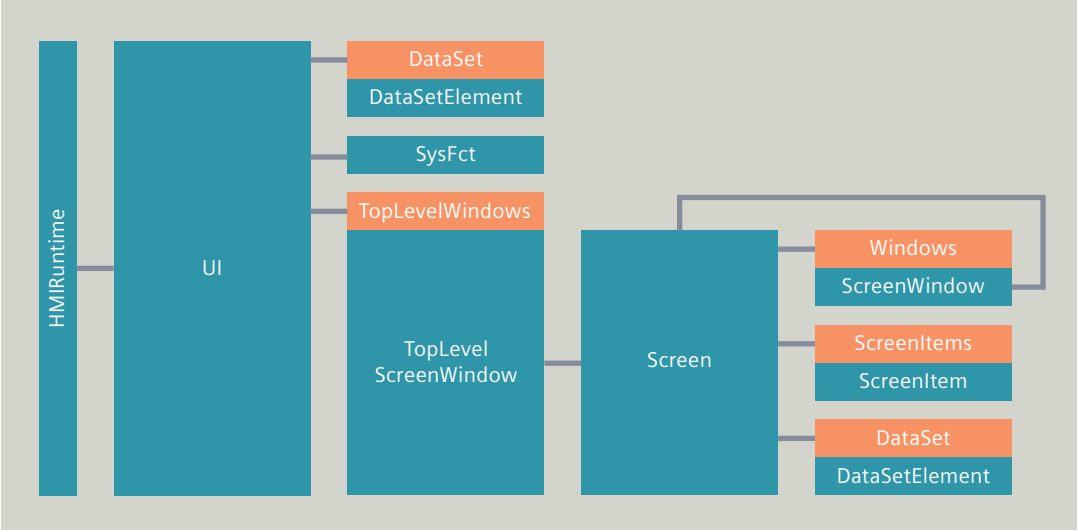
See also

Timers (Page 1390)

Timers.ClearTimeout() (Page 1392)

10.2.2.20 UI

Description



The "UI" object ("HMIUI" type) represents the user interface of the graphical runtime. You use the "UI" object to directly reference the currently active screen or the listing of the screen windows on the highest level.

Application

The "UI" object is used to reference the configured elements of the graphical runtime system, such as screen windows, screens or screen objects. This means that you have access to all the properties and methods of these elements.

To simplify the use of the "UI" object, you can also use the alias `UI` for `HMIRuntime.UI`.

Note

Several screens can be opened simultaneously in runtime. These screens are displayed in screen windows. Each screen window contains exactly one screen (Screen) that can contain any number of further screen windows (Screen Window). The resulting hierarchy can be mapped by using an object path, which is used in the "FindItem" method for addressing.

In the window arrangement of the runtime (Screen Window Layout), one screen window can be defined at the highest level (Top Level Screen Window) for each existing monitor.

Object type

HMIUI

Properties

The "UI" object has the following properties:

- **ActiveScreen**
  Returns the screen that has the input focus.
- **DataSet**
  Returns the global parameter set.
- **PopupScreenWindows**
  Returns the list of all popup screen windows.
- **RootWindow**
  Returns the top-level screen window (Top Level Screen Window) of the screen in which the script is executed.
- **Style**
  Specifies the style of the display and operating objects.
- **Windows**
  Returns the "Windows" object with the list of screen windows.

Methods

The "UI" object has the following methods:

- **FindItem()**
  Returns screen windows or screen objects through their object path.
- **GetClientInfo()**
  Returns information about the client.
- **OpenFaceplateInPopup()**
  Opens a faceplate in a popup window and sets the values of the faceplate interface.

Example

Modify the screen of the own screen window on the highest level:

```
UI.RootWindow.Screen = 'NewScreen';
```

or with the "FindItem" method and relative addressing:

```
UI.FindItem('~').Screen = 'NewScreen';
```

Reference and modify the screen of a screen window in absolute terms on the highest level outside of the own screen hierarchy:

```
UI.FindItem('/TopLevelWindow2').Screen = 'NewScreen';
```

UI.ActiveScreen

Description

The "ActiveScreen" property returns the screen that has the input focus.

Note

You access the screen with input focus via the "ActiveScreen" property without specification of an object path.

If the runtime loses the input focus, the "ActiveScreen" property references the screen that last had the input focus.

Type

Object, HmiScreen (Page 1397)

Access

Read-only

Syntax

```
[HMIRuntime.]UI.ActiveScreen
```

See also

UI (Page 1395)

Screen (Page 1397)

Screen

Description

The "Screen" object ("HMIScreen" type) displaya screen in runtime. A screen can contain any number of screen windows ("Window" objects) and screen objects ("ScreenItem" object).

You reference a "Screen" object via the properties `UI.ActiveScreen`, `Window.CurrentScreen`, `Screen.ParentScreen` or `ScreenItem.Parent`.

Use

You can use the "this" and "Item" objects in scripts. The "this" object refers in this case to the screen ("Screen" object) and the "Item" object to the screen object (ScreenItem) in which the script is created. For better clarity, you can also use the `Screen` alias for the "this" object.

Note

Several screens can be opened simultaneously in runtime. These screens are displayed in screen windows. Each screen window contains exactly one screen (Screen) that can contain any number of additional screen windows (Screen Window). The resulting hierarchy can be mapped by using an object path, which is used in the "FindItem" method for addressing.

You can specify one top level screen window (Top Level Screen Window) for each monitor in the window layout of the runtime system (Screen Window Layout).

Object type

HmiScreen

Properties

The "Screen" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the fill pattern of the background.
- **BackGraphic**
  Specifies the graphic for the image background.
- **BackGraphicStretchMode**
  Specifies the type of scaling of the background graphic in the screen.
- **BackgroundFillMode**
  Specifies the fill area of the background fill.
- **CurrentWindow**
  Returns the screen window which contains the current screen.
- **DataSet**
  Returns the parameter set of the screen.
- **DisplayName**
  Specifies the display name.
- **Enabled**
  Specifies whether the screen can be operated in runtime.

- **EnableExplicitUnlock**
  Returns which button must be pressed for the screen to be operable.
- **Height**
  Specifies the height.
- **HorizontalAlignment**
  Specifies the horizontal alignment.
- **HotKeys**
  Returns the hotkeys defined for the screen.
- **Items**
  Returns a list of all screen objects of the screen.
- **Layers**
  Returns a list of all layers of the screen.
- **Name**
  Returns the name of the screen.
- **Operability**
  Returns whether the screen is operable.
- **Parent**
  Returns the higher-level screen object (here: screen window).
- **ParentScreen**
  Returns the higher-level screen.
- **RequireExplicitUnlock**
  Returns whether the screen can only be operated while the associated button is being pressed.
- **ScreenMaster**
  Specifies the screen template of the screen.
- **ScreenNumber**
  Returns the screen number.
- **VerticalAlignment**
  Specifies the vertical alignment.
- **Width**
  Specifies the width.
- **Windows**
  Returns a list of all screen windows of the screen.

Methods

The "Screen" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the screen.
- **FindItem()**
  Returns screen windows or screen objects through their object path.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Screen" object has the following events:

- **OnContextTapped()**
  Occurs when the screen is right-clicked or long-touched.
- **OnHotKey()**
  Occurs when a hotkey is pressed.
- **OnLoaded()**
  Occurs when the screen is fully loaded in runtime.
- **OnTapped()**
  Occurs when the screen is clicked with the left mouse button or short-touched.
- **OnUnloaded()**
  Occurs when the active screen on the HMI device is completely cleared.

Screen.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Screen.AlternateBackColor
```

See also

Screen (Page 1397)

Screen.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the screen.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Screen.Authorization
```

See also

Screen (Page 1397)

Screen.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Screen.BackColor
```

See also

Screen (Page 1397)

Screen.BackFillPattern

Description

The "BackFillPattern" property specifies the fill pattern of the background.

Type

Int32, HmiFillPattern

Specifies the fill pattern:

- Solid (0): Solid
- Transparent (65536): Transparent. Depending on the runtime settings, visible objects in the background can be selected.
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonally striped
- BackwardDiagonal (131075): Backward diagonally striped
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Gradient vertical tricolor
- GradientForwardDiagonalTricolor (1048834): Gradient forward diagonal tricolor
- GradientBackwardDiagonalTricolor (1048835): Gradient backward diagonal tricolor

Access

Read-write

Syntax

```
Screen.BackFillPattern
```

See also

Screen (Page 1397)

Screen.BackGraphic

Description

The "BackGraphic" property specifies the graphic for the screen background.

Type

String

Access

Read-write

Syntax

```
Screen.BackGraphic
```

See also

Screen (Page 1397)

Screen.BackGraphicStretchMode

Description

The "BackGraphicStretchMode" property specifies the type of scaling of the background graphic in the screen.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Screen.BackGraphicStretchMode
```

See also

Screen (Page 1397)

Screen.BackgroundFillMode

Description

The "BackgroundFillMode" property specifies the fill area of the background fill.

Type

Int32, HmiBackgroundFillMode

Specifies the type of background fill:

- Window (0): The filling is adapted to the size of the window.
- Screen (1): The filling is adapted to the size of the screen.

Access

Read-write

Syntax

```
Screen.BackgroundFillMode
```

See also

Screen (Page 1397)

Screen.CurrentWindow

Description

The "CurrentWindow" property returns the screen window which contains the current screen.

Type

Object, HmiScreenWindow (Page 1436)

Access

Read-only

Syntax

```
Screen.CurrentWindow
```

See also

Screen (Page 1397)

ScreenWindow (Page 1436)

ScreenWindow

Description

ScreenWindow (Page 1436)

Screen.DataSet

Description

The "DataSet" property returns the data set ("DataSet" object) of the screen.

Type

Object, HMIDataSet (Page 1405)

Access

Read-only

Syntax

```
Screen.DataSet
```

See also

Screen (Page 1397)

DataSet (Page 1405)

DataSet

Description

The "DataSet" object ("HMIDataSet" type) is a list of "DataSetElement" objects that enable data exchange in runtime.

A "DataSet" object is defined globally (on the "UI" object) or on a screen (the "Screen" object). You can access the data from any action.

You reference a "DataSet" object via the `UI.DataSet` property or via the respective screen with the `Screen.DataSet` property.

The "DataSet" object exists as long as the screen is displayed. The global object exists over the entire time period in which runtime runs.

Use

The "DataSet" object is a list which be counted and enumerated. You can access the "DataSet" list via the index or the element names.

Object type

HMIDataSet

Properties

The "DataSet" object has the following properties:

- **Count**
  Returns the number of elements of the "DataSet" list.

Methods

The "DataSet" object has the following methods:

- **Add()**
  Adds an element with value to the "DataSet" list.
- **Clear()**
  Removes all elements from the "DataSet" list.
- **Exists()**
  Checks whether an element is present in the "DataSet" list.
- **Item()**
  Returns the value of an element of the "DataSet" list.
- **Remove()**
  Removes an element by its name from the "DataSet" list.

DataSet.Count

Description

The "Count" property returns the number of elements in the "DataSet" list.

Type

UInt32

Access

Read-only

Syntax

```
DataSet.Count
```

See also

DataSet (Page 1405)

DataSet.Add()

Description

The "Add" method adds an element ("DataSetElement" object) to the "DataSet" list. The elements are referenced by name.

Syntax

```
DataSet.Add(name,value)
```

Parameters

name

Type: String

Name of the element that will be added to the list.

value

Type: Variant

Value of the element that will be added to the list.

Return value

--

See also

DataSet (Page 1405)

DataSet.Clear()

Description

The "Clear" method removes all elements ("DataSetElement" objects) from the "DataSet" list.

Syntax

```
DataSet.Clear()
```

Parameter

--

Return value

--

See also

DataSet (Page 1405)

DataSet.Exists()

Description

The "Exists" method searches for an element ("DataSetElement" object) in the "DataSet" list.

Syntax

```
DataSet.Exists(name)
```

Parameters

name

Type: String

Name of the element that is being searched for.

Return value

Bool

See also

DataSet (Page 1405)

DataSet.Item()

Description

The "Item" method returns the value of a "DataSetElement" object of the "DataSet" list.

Syntax

```
DataSet[.Item](name)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "DataSet" object.

Parameter

`name`

Type: String

Element name or index number (1...n) of an object "DataSetElement" of the list

Return value

Variant

See also

DataSet (Page 1405)

DataSetElement (Page 1410)

DataSet.Remove()

Description

The "Remove" method removes an element ("DataSetElement" object) from the "DataSet" list by its name.

Syntax

```
DataSet.Remove(name)
```

Parameters

`name`

Type: String

Name of the element that will be removed.

Return value

--

See also

DataSet (Page 1405)

DataSetElement

Description

The "DataSetElement" object represents a data set element in the "DataSet" list.

Object type

HMIDataSetElement

Properties

The "DataSetElement" object has the following properties:

- **Name**
  Returns the name of the element.
- **Value**
  Returns the value of the element.

Methods

--

DataSetElement.Name

Description

The "Name" property returns the name of the element ("DataSetElement" object).

Type

String

Access

Read-only

Syntax

```
DataSetElement.Name
```

See also

DataSetElement (Page 1410)

DataSetElement.Value

Description

The "Name" property returns the value of the element ("DataSetElement" object).

Type

Variant

Access

Read-only

Syntax

```
DataSetElement.Value
```

See also

DataSetElement (Page 1410)

Screen.DisplayName

Description

The "DisplayName" property specifies the display name of the screen.

Type

String

Access

Read-write

Syntax

```
Screen.DisplayName
```

See also

Screen (Page 1397)

Screen.Enabled

Description

The "Enabled" property specifies whether the screen can be operated with all screen objects in runtime.

Type

Bool

Access

Read-write

Syntax

```
Screen.Enabled
```

See also

Screen (Page 1397)

Screen.EnableExplicitUnlock

Description

The "EnableExplicitUnlock" property returns the button that must be pressed for the screen to be operable.

Type

Object, HmiButton (Page 1961)

Access

Read-only

Syntax

```
Screen.EnableExplicitUnlock
```

See also

Screen (Page 1397)

Button (Page 1961)

Button

Description

Button (Page 1961)

Screen.Height

Description

The "Height" property specifies the height of the screen in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Screen.Height
```

See also

Screen (Page 1397)

Screen.HorizontalAlignment

Description

The "HorizontalAlignment" property specifies the horizontal alignment of the screen. This property is used when the screen is smaller than the screen window.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Screen.HorizontalAlignment
```

See also

Screen (Page 1397)

Screen.Items

Description

The "Items" property returns a list of all screen objects of the screen.

Type

Object, HMIScreenItems (Page 1415)

Access

Read-only

Syntax

```
Screen.Items
```

See also

Screen (Page 1397)

ScreenItems (Page 1415)

ScreenItems

Description

The "ScreenItems" object is a list of all screen objects ("HmiScreenItemBase" objects) of the screen.

You reference a "ScreenItems" object via the `Screen.Items` property.

Use

The "ScreenItems" object is a list and can be enumerated. You can access the "ScreenItems" list using the index or the tag name.

Object type

HMIScreenItems

Properties

--

Methods

The "ScreenItems" object has the following methods:

- **Item()**
  Returns a screen object of the "ScreenItems" list.

ScreenItems.Item()

Description

The "Item" method returns a screen object of a screen window via the "ScreenItems" list.

Syntax

```
ScreenItems[.Item](ScreenItemName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "ScreenItems" object.

Parameters

`ScreenItemName`
Type: String, HmiScreenItemBase
Name of the screen object

Return value

Object, HmiScreenItemBase (Page 1571)

See also

ScreenItems (Page 1415)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Screen.Layers

Description

The "Layers" property returns a list of all layers of the screen.

Type

Object, HmiLayerCollection (Page 1417)

Access

Read-only

Syntax

```
Screen.Layers
```

See also

Screen (Page 1397)
HmiLayerCollection (Page 1417)

HmiLayerCollection

Description

The "HmiLayerCollection" object is a list of all layers ("Layer" objects) of the screen.

You reference a "HmiLayerCollection" object via the `Screen.Layers` property.

Use

The "HmiLayerCollection" object is a list and can be counted and enumerated. You can access the "HmiLayerCollection" list using the index or the tag name.

Object type

HmiLayerCollection

Properties

The "HmiLayerCollection" object has the following properties:

- **Count**
  Returns the number of layers of the "HmiLayerCollection" list.

Methods

The "HmiLayerCollection" object has the following methods:

- **Item()**
  Returns a level of the "HmiLayerCollection" list.

HmiLayerCollection.Count

Description

The "Count" property returns the number of screen layers of the "HmiLayerCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiLayerCollection.Count
```

See also

HmiLayerCollection (Page 1417)

HmiLayerCollection.Item()

Description

The "Item" method returns a screen layer of the "HmiLayerCollection" list.

Syntax

```
HmiLayerCollection[.Item](HmiLayerName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiLayerCollection" object.

Parameters

`HmiLayerName`

Type: String, HmiLayerPart

Name of the screen layer

Return value

Object, HmiLayerPart (Page 1418)

See also

HmiLayerCollection (Page 1417)

Layer (Page 1418)

Layer

Description

The "Layer" object represents the layers of an object (e.g. screen).

Object type

HmiLayerPart

Properties

The "Layer" object has the following properties:

- **MaximumZoom**
  Specifies the maximum zoom of the screen up to which the level is to be seen.
- **MinimumZoom**
  Specifies the minimum zoom of the screen up to which the level is to be seen.
- **Name**
  Returns the name of the screen layer.
- **Visible**
  Specifies whether the screen layer and contained objects are visible.

Methods

--

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Layer (Page 1418)
Layer.MinimumZoom (Page 1419)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Layer (Page 1418)
Layer.MaximumZoom (Page 1419)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Layer (Page 1418)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Layer (Page 1418)

Screen.Name

Description

The "Name" property returns the name of the screen.

Type

String

Access

Read-only

Syntax

```
Screen.Name
```

See also

Screen (Page 1397)

Screen.Operability

Description

The "Operability" property returns whether the screen is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Screen.Operability
```

See also

Screen (Page 1397)

Screen.Parent

Description

The "Parent" property specifies the parent screen object (here: screen window).

Type

Object, HmiScreenWindow (Page 1436)

Access

Read-only

Syntax

```
Screen.Parent
```

See also

Screen (Page 1397)

ScreenWindow (Page 1436)

ScreenWindow

Description

ScreenWindow (Page 1436)

Screen.ParentScreen

Description

The "ParentScreen" property returns the parent screen that contains the screen window of the current screen.

Type

Object, HmiScreen (Page 1397)

Access

Read-only

Syntax

```
Screen.ParentScreen
```

See also

Screen (Page 1397)

Screen

Description

Screen (Page 1397)

Screen.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the screen can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Screen.RequireExplicitUnlock
```

See also

Screen (Page 1397)

Screen.ScreenMaster

Description

The "ScreenMaster" property specifies the screen template of the screen. All screen objects, layers and properties of the referenced screen template are inherited.

Type

Object, HmiScreenMaster (Page 1424)

Access

Read-write

Syntax

```
Screen.ScreenMaster
```

See also

Screen (Page 1397)

ScreenMaster (Page 1424)

ScreenMaster

Description

The "ScreenMaster" object is the screen template of a screen. All screen objects, layers and properties of a screen template are inherited by the screen. The screen template is a type of main layer of the screen.

Object type

HmiScreenMaster

Properties

The "ScreenMaster" object has the following properties:

- **DisplayName**
  Specifies the display name.
- **HotKeys**
  Returns the hotkeys specified for the screen template.
- **Layers**
  Returns a list of all layers of the screen template.
- **Name**
  Returns the name of the screen template.
- **Visible**
  Specifies whether the screen template is visible in the screen.

Methods

--

ScreenMaster.DisplayName

Description

The "DisplayName" property specifies the display name of the screen template.

Type

String

Access

Read-write

Syntax

```
ScreenMaster.DisplayName
```

See also

ScreenMaster (Page 1424)

ScreenMaster.Layers

Description

The "Layers" property returns a list of all layers of the screen template.

Type

Object, HmiLayerCollection (Page 1426)

Access

Read-only

Syntax

```
ScreenMaster.Layers
```

See also

ScreenMaster (Page 1424)

HmiLayerCollection (Page 1426)

HmiLayerCollection

Description

The "HmiLayerCollection" object is a list of all layers ("Layer" objects) of the screen.

You reference a "HmiLayerCollection" object via the `Screen.Layers` property.

Use

The "HmiLayerCollection" object is a list and can be counted and enumerated. You can access the "HmiLayerCollection" list using the index or the tag name.

Object type

HmiLayerCollection

Properties

The "HmiLayerCollection" object has the following properties:

- **Count**
  Returns the number of layers of the "HmiLayerCollection" list.

Methods

The "HmiLayerCollection" object has the following methods:

- **Item()**
  Returns a level of the "HmiLayerCollection" list.

HmiLayerCollection.Count

Description

The "Count" property returns the number of screen layers of the "HmiLayerCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiLayerCollection.Count
```

See also

HmiLayerCollection (Page 1426)

HmiLayerCollection.Item()

Description

The "Item" method returns a screen layer of the "HmiLayerCollection" list.

Syntax

```
HmiLayerCollection[.Item](HmiLayerName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiLayerCollection" object.

Parameters

`HmiLayerName`

Type: String, HmiLayerPart

Name of the screen layer

Return value

Object, HmiLayerPart (Page 1428)

See also

HmiLayerCollection (Page 1426)

Layer (Page 1428)

Layer

Description

The "Layer" object represents the layers of an object (e.g. screen).

Object type

HmiLayerPart

Properties

The "Layer" object has the following properties:

- **MaximumZoom**
  Specifies the maximum zoom of the screen up to which the level is to be seen.
- **MinimumZoom**
  Specifies the minimum zoom of the screen up to which the level is to be seen.
- **Name**
  Returns the name of the screen layer.
- **Visible**
  Specifies whether the screen layer and contained objects are visible.

Methods

--

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Layer (Page 1428)

Layer.MinimumZoom (Page 1429)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Layer (Page 1428)

Layer.MaximumZoom (Page 1429)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Layer (Page 1428)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Layer (Page 1428)

ScreenMaster.Name

Description

The "Name" property returns the name of the screen template.

Type

String

Access

Read-only

Syntax

```
ScreenMaster.Name
```

See also

ScreenMaster (Page 1424)

ScreenMaster.Visible

Description

The "Visible" property specifies whether the screen template is visible. The screen template is a type of main layer of a screen.

Type

Bool

Access

Read-write

Syntax

```
ScreenMaster.Visible
```

See also

ScreenMaster (Page 1424)

Screen.ScreenNumber

Description

The "ScreenNumber" property returns the screen number.

Type

UInt16

Access

Read-only

Syntax

```
Screen.ScreenNumber
```

See also

Screen (Page 1397)

Screen.VerticalAlignment

Description

The "VerticalAlignment" property specifies the vertical alignment.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Screen.VerticalAlignment
```

See also

Screen (Page 1397)

Screen.Width

Description

The "Width" property specifies the width of the screen in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Screen.Width
```

See also

Screen (Page 1397)

Screen.Windows

Description

The "Windows" returns a list of all screen windows of the screen.

Type

Object, HMIWindows (Page 1434)

Access

Read-only

Syntax

```
Screen.Windows
```

See also

Screen (Page 1397)

Windows (Page 1434)

Windows

Description

The "Windows" object is a list of all screen windows ("ScreenWindow" objects) of the screen.

You reference a "Windows" object via the `Screen.Windows` property.

Use

The "Windows" object is a list and can be counted and enumerated. You can access the "Windows" list using the index or the tag name.

Object type

HMIWindows

Properties

The "Windows" object has the following properties:

- **Count**
  Returns the number of screen windows of the "Windows" list.

Methods

The "Windows" object has the following methods:

- **Item()**
  Returns a screen window of the "Windows" list.

Windows.Count

Description

The "Count" property returns the number of screen windows in the "Windows" list.

Type

UInt32

Access

Read-only

Syntax

```
Windows.Count
```

See also

Windows (Page 1434)

Windows.Item()

Description

The "Item" method returns a screen window of the "Windows" list.

Syntax

```
Windows[.Item](WindowName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "Windows" object.

Parameters

`WindowName`

Type: String, HmiScreenWindow

Name of the screen window

Return value

Object, HmiScreenWindow (Page 1436)

See also

Windows (Page 1434)

ScreenWindow (Page 1436)

ScreenWindow

Description

The "ScreenWindow" object represents a screen window in runtime. A screen window contains exactly one screen ("Screen" object).

The "ScreenWindow" object is returned from the `Screen.Windows` list, the `FindItem()` method or the `Screen.CurrentWindow` and `Screen.Parent` properties.

Object type

HmiScreenWindow

Use

You use the "ScreenWindow" object to reference a "Screen" object and have access to all the objects and properties of a screen.

Note

Several screens can be opened simultaneously in runtime. These screens are displayed in screen windows. Each screen window contains exactly one screen (Screen) that can contain any number of additional screen windows (Screen Window). The resulting hierarchy can be mapped by using an object path, which is used in the "FindItem" method for addressing.

You can specify one top level screen window (Top Level Screen Window) for each monitor in the window layout of the runtime system (Screen Window Layout).

Properties

The "ScreenWindow" object has the following properties:

- **Adaption**
  Specifies how the window size adapts.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the specified screen window.
- **CurrentScreen**
  Returns the screen of the current screen window.
- **CurrentZoomFactor**
  Specifies the zoom factor which is applied to the displayed screen.
- **Enabled**
  Specifies whether the specified object can be operated in runtime.

- **Height**
  Specifies the height.
- **HorizontalScrollBarPosition**
  Specifies the horizontal alignment for the scroll bar.
- **HorizontalScrollBarVisibility**
  Specifies the setting for the horizontal scroll bar of the window.
- **Icon**
  Specifies the icon.
- **InteractiveZooming**
  Specifies whether zooming is supported.
- **Layer**
  Returns the layer of the screen that contains the screen window.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin of the screen window.
- **Monitor**
  Returns the monitor on which the screen window is displayed.
- **Name**
  Returns the name of the screen window.
- **Parent**
  Returns the higher-level screen object (Parent container).
- **Path**
  Returns the absolute object path of the screen window.
- **RenderingTemplate**
  Returns the name of the template from which the object was created.
- **Screen**
  Specifies the name of the screen contained in the screen window.
- **ScreenName**
  Returns the screen name.
- **ScreenNumber**
  Returns the screen number.
- **ShowFocusVisual**
  Specifies whether the screen window is highlighted when in focus.
- **StartupPosition**
  Specifies the position of the screen window at runtime start.
- **StyleItemClass**
  Returns the style which is applied to the screen window.
- **System**
  Specifies the server prefix.
- **TabIndex**
  Returns the position of the screen window in the tab sequence.

- **TabIntoWindow**
  Specifies that the configured tab sequence of the displayed screen is resumed on activation via the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **VerticalScrollBarPosition**
  Specifies the vertical alignment for the scroll bar.
- **VerticalScrollBarVisibility**
  Specifies the setting for the vertical scroll bar of the window.
- **Visible**
  Specifies whether the screen window is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the screen window configuration.

Methods

The "ScreenWindow" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the screen window.
- **Close()**
  Closes the screen window.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ScreenWindow" object has the following events:

- **OnActivated()**
  Occurs when a screen window receives focus.
- **OnDeactivated()**
  Occurs when a screen window loses focus.

Example

Change the screen in the adjacent screen window "Window2":

```
Screen.ParentScreen.Windows('Window2').Screen = 'NewScreen';
```

or with the "FindItem" method and relative addressing:

```
Screen.FindItem('../Window2').Screen = 'NewScreen';
```

ScreenWindow.Adaption

Description

The "Adaption" property specifies how the window size adapts.

Type

Int32, HmiScreenWindowAdaption

Specifies how the window size adapts:

- None (0): No adaptation
- WindowToScreen (1): Window size corresponds to screen size
- ScreenToWindow (2): Screen is scaled to window size.

Access

Read-write

Syntax

```
ScreenWindow.Adaption
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ScreenWindow.Caption
```

See also

ScreenWindow (Page 1436)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ScreenWindow.Caption (Page 1439)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 1440)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 1440)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 1440)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 1440)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 1440)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 1440)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ScreenWindow.Caption (Page 1439)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ScreenWindow.Caption (Page 1439)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ScreenWindow.Caption (Page 1439)

ScreenWindow.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ScreenWindow.CaptionColor
```

See also

ScreenWindow (Page 1436)

ScreenWindow.CurrentQuality

Description

The property "CurrentQuality" returns the current worst quality code of all tags which influence the specified screen window.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable. Quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ScreenWindow.CurrentQuality
```

See also

ScreenWindow (Page 1436)

ScreenWindow.CurrentScreen

Description

The "CurrentScreen" property returns the screen of the current screen window.

Type

Object, HmiScreen (Page 1397)

Access

Read-only

Syntax

```
ScreenWindow.CurrentScreen
```

See also

Screen (Page 1397)

ScreenWindow (Page 1436)

Screen

Description

Screen (Page 1397)

ScreenWindow.CurrentZoomFactor

Description

The "CurrentZoomFactor" property specifies the zoom factor of the screen window. The zoom factor may differ from the containing screen. The value 1.0 corresponds to a zoom factor of 100%.

Type

Float

Access

Read-write

Syntax

```
ScreenWindow.CurrentZoomFactor
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Enabled

Description

The "Enabled" property specifies whether the screen window can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ScreenWindow.Enabled
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Height

Description

The "Height" property specifies the height of the screen window in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ScreenWindow.Height
```

See also

ScreenWindow (Page 1436)

ScreenWindow.HorizontalScrollBarPosition

Description

The "HorizontalScrollBarPosition" property specifies the horizontal alignment for the scroll bar.

Type

Int32

Access

Read-write

Syntax

```
ScreenWindow.HorizontalScrollBarPosition
```

See also

ScreenWindow (Page 1436)

ScreenWindow.HorizontalScrollBarVisibility (Page 1449)

ScreenWindow.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
ScreenWindow.HorizontalScrollBarVisibility
```

See also

ScreenWindow (Page 1436)

ScreenWindow.HorizontalScrollBarPosition (Page 1448)

ScreenWindow.Icon

Description

The "Icon" property specifies the icon of the screen window.

Type

String

Access

Read-write

Syntax

```
ScreenWindow.Icon
```

See also

ScreenWindow (Page 1436)

ScreenWindow.InteractiveZooming

Description

The "InteractiveZooming" property specifies whether zooming is supported.

Type

Bool

Access

Read-write

Syntax

```
ScreenWindow.InteractiveZooming
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Layer

Description

The "Layer" property returns the layer of the screen that contains the screen window.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ScreenWindow.Layer
```

See also

ScreenWindow (Page 1436)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ScreenWindow.Layer (Page 1451)

Layer.MinimumZoom (Page 1452)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ScreenWindow.Layer (Page 1451)
Layer.MaximumZoom (Page 1451)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ScreenWindow.Layer (Page 1451)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ScreenWindow.Layer (Page 1451)

ScreenWindow.Left

Description

The "Left" property specifies the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ScreenWindow.Left
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Margin

Description

The "Margin" property specifies the surrounding outer distance of the screen window.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ScreenWindow.Margin
```

See also

ScreenWindow (Page 1436)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ScreenWindow.Margin (Page 1454)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ScreenWindow.Margin (Page 1454)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ScreenWindow.Margin (Page 1454)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ScreenWindow.Margin (Page 1454)

ScreenWindow.Monitor

Description

The "Monitor" property returns the monitor on which the window is displayed.

Type

UInt8

Access

Read-only

Syntax

```
ScreenWindow.Monitor
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Name

Description

The "Name" property returns the name of the screen window.

Type

String

Access

Read-only

Syntax

```
ScreenWindow.Name
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Parent

Description

The "Parent" property returns the parent screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ScreenWindow.Parent
```

See also

ScreenWindow (Page 1436)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ScreenWindow.Path

Description

The "Path" property returns the absolute object path of a screen window in runtime starting from the screen window on the highest level.

Note

For the syntax of an object path, see the "FindItem" method (Page 1549).

Type

String

Access

Read-only

Syntax

```
ScreenWindow.Path
```

See also

UI.FindItem() (Page 1549)

ScreenWindow (Page 1436)

ScreenWindow.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the object has been created.

Type

String

Access

Read-only

Syntax

```
ScreenWindow.RenderingTemplate
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Screen

Description

The "Screen" property specifies the name of the screen ("HMIScreen" type) that is contained in the screen window. Loads a new screen via its name into the screen window.

The "Screen" property returns a different value than the "CurrentScreen" when the referenced screen is not yet loaded completely or does not exist.

Type

String, HmiStoredScreen

Access

Read-write

Syntax

```
ScreenWindow.Screen
```

See also

ScreenWindow (Page 1436)

ScreenWindow.ScreenName

Description

The "ScreenName" property returns the screen name.

Type

String

Access

Read-only

Syntax

```
ScreenWindow.ScreenName
```

See also

ScreenWindow (Page 1436)

ScreenWindow.ScreenNumber

Description

The "ScreenNumber" property returns the screen number.

Type

UInt16

Access

Read-only

Syntax

```
ScreenWindow.ScreenNumber
```

See also

ScreenWindow (Page 1436)

ScreenWindow.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the screen window is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ScreenWindow.ShowFocusVisual
```

See also

ScreenWindow (Page 1436)

ScreenWindow.StartupPosition

Description

The "StartupPosition" property sets the position of the screen window at runtime start.

Type

Int32, HmiWindowStartupPosition

Specifies the position of the screen window:

- None (0): Relative placement on the configured monitor via "Left" and "Top".
- CenteredMonitor (1): Centered on the configured monitor.
- Maximized (2): Maximized on the configured monitor.
- CenteredOwner (3): Centered on the displayed screen.

Access

Read-write

Syntax

```
ScreenWindow.StartupPosition
```

See also

ScreenWindow (Page 1436)

ScreenWindow.StyleItemClass

Description

The "StyleItemClass" property returns the style that is applied to the screen window.

Type

String

Access

Read-only

Syntax

```
ScreenWindow.StyleItemClass
```

See also

ScreenWindow (Page 1436)

ScreenWindow.System

Description

The "System" property specifies the server prefix.

Type

String, HmiSystem

Access

Read-write

Syntax

```
ScreenWindow.System
```

See also

ScreenWindow (Page 1436)

ScreenWindow.TabIndex

Description

The "TabIndex" returns the position of the screen window in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ScreenWindow.TabIndex
```

See also

ScreenWindow (Page 1436)

ScreenWindow.TabIntoWindow

Description

The "TabIntoWindow" property specifies that the configured tab sequence of the displayed screen is resumed on activation via the configured tab sequence.

Type

Bool

Access

Read-write

Syntax

```
ScreenWindow.TabIntoWindow
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Top

Description

The "Top" property specifies the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ScreenWindow.Top
```

See also

ScreenWindow (Page 1436)

ScreenWindow.VerticalScrollBarPosition

Description

The "VerticalScrollBarPosition" property specifies the vertical position for the scroll bar.

Type

Int32

Access

Read-write

Syntax

```
ScreenWindow.VerticalScrollBarPosition
```

See also

ScreenWindow (Page 1436)

ScreenWindow.VerticalScrollBarVisibility (Page 1464)

ScreenWindow.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the window.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
ScreenWindow.VerticalScrollBarVisibility
```

See also

ScreenWindow (Page 1436)

ScreenWindow.VerticalScrollBarPosition (Page 1464)

ScreenWindow.Visible

Description

The "Visible" property specifies whether the screen window is visible.

Type

Bool

Access

Read-write

Syntax

```
ScreenWindow.Visible
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Width

Description

The "Width" property specifies the width of the screen window in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ScreenWindow.Width
```

See also

ScreenWindow (Page 1436)

ScreenWindow.WindowFlags

Description

The "WindowFlags" property specifies the screen window configuration.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
ScreenWindow.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

ScreenWindow (Page 1436)

ScreenWindow.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the screen window.

Syntax

```
ScreenWindow.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ScreenWindow (Page 1436)

ScreenWindow.Close()

Description

The "Close" method closes the screen window.

Syntax

```
ScreenWindow.Close()
```

Parameters

--

Return value

Bool

See also

ScreenWindow (Page 1436)

ScreenWindow.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing means the change between two values of a property.

Syntax

```
ScreenWindow.PropertyFlashing(propertyName,enable[,value]
[,alternateValue][,rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flashing frequency:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ScreenWindow (Page 1436)

ScreenWindow_OnActivated()

Description

The "OnActivated" event occurs when a screen window receives focus:

- A screen window is selected via the configured tab sequence.
- A screen window that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and receives focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
ScreenWindow_OnActivated(item)
```

Context

`Item`

Type: Object

Screen window where the event occurs.

See also

ScreenWindow (Page 1436)

ScreenWindow_OnDeactivated()

Description

The "OnDeactivated" event occurs when a screen window loses focus when the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
ScreenWindow_OnDeactivated(item)
```

Context

`Item`

Type: Object

Screen window where the event occurs.

See also

ScreenWindow (Page 1436)

Screen.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the screen.

Syntax

```
Screen.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Screen (Page 1397)

Screen.FindItem()

Description

The "FindItem" method returns screen windows or screen objects through their object path.

Syntax

```
Screen.FindItem(ScreenItemPath)
```

Parameter

`ScreenItemPath`

Type: String

Object path of the searched screen window or screen object.

Note

The "UI.FindItem" method has a global search context and requires absolute object paths. The "Screen.FindItem" method has the current screen as the search context and can also use relative object paths.

Formulation of the object path

The syntax of the object path orients itself to the notation of the file system paths. The object path consists of the names of the screen windows (Screen Windows) and screen objects (Screen Items). The names are connected via a slash ("/") according to the hierarchical positioning. Screens (Screens) and their names are not used in the formulation.

Relative and absolute object paths are distinguished by the prefix of the object path. The following prefixes can be used:

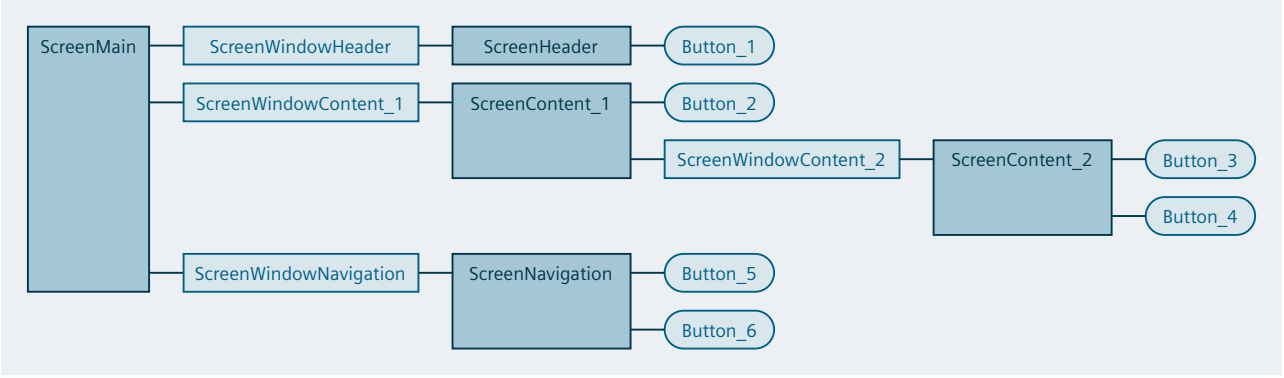
- Relative object path
  - "..": References the parent screen window (Parent) in the context of the current screen window.
  - ".": References the own screen window (Self).
  - "": A screen object of the current screen window is referenced without prefix.
- Absolute object path
  - "/": References a screen window on the highest level, whose name must follow.
  - "~": References the screen window on the highest level in the own screen hierarchy.

Additional rules for formulating an object path:

- The string ".." may be used several times in the object path, but only together at the beginning of the object path, for example, "../../Window5".
- If the object path does not end with a screen object name, a screen window is referenced.
- A search is performed for screen objects of the object path in the screens of the referenced screen window. It is not permitted to specify a screen name.

Examples of object paths

The following window / screen object hierarchy is adopted for the following examples:



The following objects paths for addressing the object result from this:

- Relative addressing:
  - "Button_2" changes the label of "Button_1":
    ```
    // Navigate one level up and find "Button_1" inside the
    "ScreenWindowHeader"
    Screen.FindItem("../ScreenWindowHeader/Button_1").Text =
    "Changed by Button_2"
    ```
  - "Button_3" changes the label of "Button_5":
    ```
    // Navigate two levels up and find "Button 2" inside the
    "ScreenWindowNavigation"
    Screen.FindItem("../../ScreenWindowNavigation/Button_5").Text =
    "Changed by Button_3"
    ```
  - "Button_3" changes the label of "Button_4":
    ```
    // Find "Button_4" in same screen ("ScreenContent_2")
    Screen.FindItem("Button_4").Text = "Changed by Button_3"
    Screen.FindItem("./Button_4").Text = "Changed by Button_3"
    ```
- Absolute addressing:
  - "Button_4" changes the label of "Button_6":
    ```
    // Navigate to the root screen and find "Button_6" inside the
    "ScreenWindowNavigation"
    Screen.FindItem("~/ScreenWindowNavigation/Button_6").Text =
    "Changed by Button_4"
    ```

Return value

Object, HmiScreenObjectBase (Page 1571)

See also

Screen (Page 1397)

Screen Items (Page 1571)

Screen.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing means the change between two values of a property.

Syntax

```
Screen.PropertyFlashing(propertyName,enable[,value][,alternateValue]
[,rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flashing frequency:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Screen (Page 1397)

Screen_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- Screen is clicked with the right mouse button.
- Screen is long-touched.

  Note

  The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Screen_OnContextTapped(item,x,y,modifiers,trigger)
```

Context

Item

Type: Object

Image where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Screen (Page 1397)

Screen_OnHotKey()

Description

The "OnHotKey" event occurs when the operator presses a hotkey:

Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Syntax

```
Screen_OnHotKey(item,keyCode,modifiers)
```

Context

item

Type: Object

Image where the event occurs.

keyCode

Type: DInt

Numeric identifier of the pressed hotkey

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Screen (Page 1397)

Screen_OnLoaded()

Description

The "OnLoaded" event occurs when the screen is fully loaded in runtime.

Syntax

```
Screen_OnLoaded(item)
```

Context

Item

Type: Object

Image at which the event occurs.

See also

Screen (Page 1397)

Screen_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- Screen is clicked with the left mouse button.
- <RETURN> or <SPACE> key is pressed when a screen has the focus.
- Screen is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Screen_OnTapped(item,x,y,modifiers,trigger)
```

Context

Item

Type: Object

Image where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Screen (Page 1397)

Screen_OnUnloaded()

Description

The "OnUnloaded" event occurs when the active screen on the HMI device is completely removed.

Syntax

```
Screen_OnUnloaded(item)
```

Context

`Item`

Type: Object

Image at which the event occurs.

See also

Screen (Page 1397)

UI.Alarm

Description

The "Alarm" object enables access to the properties of a selected alarm in the alarm control.

Object type

HMIUIAlarm

Properties

--

Methods

The "Alarm" object has the following methods:

- **GetSelectedAlarmAttributes()**
  Returns all properties of a selected alarm in the alarm control.

Alarm.GetSelectedAlarmAttributes()

Description

The "GetSelectedAlarmAttributes" method returns all properties of a selected alarm in the alarm control.

This method executes an asynchronous read operation without blocking further script execution. In doing so, the method uses a promise object which has handlers for successful ("then()") and faulty ("catch()") execution of the read operation. Depending on the result, once execution is complete the corresponding handler of the promise pattern is called with the "AlarmResult" objects or the error code as parameter.

Syntax

```
[HMIRuntime.]UI.Alarm.GetSelectedAlarmAttributes(PathAlarmControl)
.then(function(HMIAlarmBlockResult[]) {
    ...
});
.catch(function(ErrorCode) {
    ...
});
```

Parameters

PathAlarmControl

Type: String, HmiAlarmControl

Path to the alarm control

Return value

Promise

Depending on the state of the promise object:

- Promise fulfilled
  Object, HMIAlarmBlockResult[] (Page 1481) as parameter of the "then()" handler.
- Promise rejected
  ErrorCode as parameter of the "catch()" handler.

See also

UI.Alarm (Page 1480)

AlarmBlockResult (Page 1481)

AlarmBlockResult

Description

The "AlarmBlockResult" object enables access to the properties of a selected alarm. The "AlarmBlockResult" object is a pure data object which maps all the properties of a selected alarm.

Use

The "AlarmBlockResult" object only contains properties and no methods.

All texts of the "AlarmBlockResult" object are monolingual strings. The language is specified with the "HMIRuntime.Language" property.

Object type

HMIAlarmBlockResult

Properties

The "AlarmBlockResult" object has the following properties:

- **AcknowledgeTime**
  Returns the time of the alarm acknowledgment.
- **AlarmGroupID**
  Returns the alarm group ID.
- **AlarmState**
  Returns the status of the alarm.
- **AlarmText1**
  Returns the localized additional text 1 of the alarm.
- **AlarmText2**
  Returns the localized additional text 2 of the alarm.
- **AlarmText3**
  Returns the localized additional text 3 of the alarm.
- **AlarmText4**
  Returns the localized additional text 4 of the alarm.
- **AlarmText5**
  Returns the localized additional text 5 of the alarm.
- **AlarmText6**
  Returns the localized additional text 6 of the alarm.
- **AlarmText7**
  Returns the localized additional text 7 of the alarm.
- **AlarmText8**
  Returns the localized additional text 8 of the alarm.
- **AlarmText9**
  Returns the localized additional text 9 of the alarm.
- **Area**
  Returns the alarm area of origin.
- **AverageActiveActive**
  Returns the result of the alarm statistics, stating how many active alarms were active on average.
- **AverageActiveAcknowledged**
  Returns the result of the alarm statistics, stating how many active alarms were acknowledged on average.
- **AverageActiveInactive**
  Returns the result of the alarm statistics, stating how many active alarms were inactive on average.
- **Class**
  Returns the name of the alarm class.

- **ClassSymbol**
  Returns the icon of the alarm class.
- **ClearTime**
  Returns the time of the alarm reset.
- **Duration**
  Returns the duration of the alarm.
- **EventText**
  Returns a localized text describing an event to the alarm.
- **Frequency**
  Returns the result of the alarm statistics, stating how often the alarm occurred.
- **HostName**
  Returns the name of the PC on which the alarm was triggered.
- **ID**
  Returns the ID of the alarm that is also used in the display.
- **InfoText**
  Returns the text information.
- **LoopInAlarm**
  Returns the name of the function that navigates from the display of the alarm to its origin.
- **ModificationTime**
  Returns the time stamp of the last alarm state modification.
- **Name**
  Returns the name of the alarm.
- **Origin**
  Returns the origin of the alarm.
- **Priority**
  Returns the relevance of the alarm or of the machine status.
- **ProcessValue1**
  Returns process value 1 of the alarm.
- **ProcessValue2**
  Returns process value 2 of the alarm.
- **ProcessValue3**
  Returns process value 3 of the alarm.
- **ProcessValue4**
  Returns process value 4 of the alarm.
- **ProcessValue5**
  Returns process value 5 of the alarm.
- **ProcessValue6**
  Returns process value 6 of the alarm.
- **ProcessValue7**
  Returns process value 7 of the alarm.
- **ProcessValue8**
  Returns process value 8 of the alarm.

- **ProcessValue9**
  Returns process value 9 of the alarm.
- **ProcessValue10**
  Returns process value 10 of the alarm.
- **RaiseTime**
  Returns the raise time of the alarm.
- **ResetTime**
  Returns the time of the alarm reset.
- **SumActiveActive**
  Returns the result of the alarm statistics, stating how many active alarms are active in total.
- **SumActiveAcknowledged**
  Returns the result of the alarm statistics, stating how many active alarms were acknowledged in total.
- **SumActiveInactive**
  Returns the result of the alarm statistics, stating how many active alarms are inactive in total.
- **SuppressionState**
  Returns the visibility status of the alarm.
- **UserName**
  Returns the name of the user who triggered the alarm.
- **Value**
  Returns the process value of the alarm.
- **ValueLimit**
  Returns the limit of the process value of the alarm.
- **ValueQuality**
  Returns the quality level of the process value of the alarm.

Methods

--

See also

Alarm.GetSelectedAlarmAttributes() (Page 1480)

AlarmBlockResult.AcknowledgeTime

Description

The "AcknowledgeTime" property returns the time of the alarm acknowledgment.

Type

DateTime

Access

Read-only

Syntax

```
AlarmBlockResult.AcknowledgeTime
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmGroupID

Description

The "AlarmGroupID" property returns the alarm group ID.

Type

UInt8

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmGroupID
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmState

Description

The "AlarmState" property returns the status of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmState
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText1

Description

The "AlarmText1" property returns the localized additional text 1 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText1
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText2

Description

The "AlarmText2" property returns the localized additional text 2 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText2
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText3

Description

The "AlarmText3" property returns the localized additional text 3 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText3
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText4

Description

The "AlarmText4" property returns the localized additional text 4 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText4
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText5

Description

The "AlarmText5" property returns the localized additional text 5 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText5
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText6

Description

The "AlarmText6" property returns the localized additional text 6 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText6
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText7

Description

The "AlarmText7" property returns the localized additional text 7 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText7
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText8

Description

The "AlarmText8" property returns the localized additional text 8 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText8
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AlarmText9

Description

The "AlarmText9" property returns the localized additional text 9 of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AlarmText9
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Area

Description

The "Area" property specifies the alarm area of origin.

The "Area" property can be configured and, together with the "Origin" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

The "Area" property, for example, includes subsystem, application name, or PLC ID. You can sort and filter alarms using the "Area" context.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.Area
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AverageActiveActive

Description

The "AverageActiveActive" property returns the result of the alarm statistics, stating how many active alarms are active on average.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AverageActiveActive
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AverageActiveAcknowledged

Description

The "AverageActiveAcknowledged" property returns the result of the alarm statistics, stating how many active alarms were acknowledged on average.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AverageActiveAcknowledged
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.AverageActiveInactive

Description

The "AverageActiveInactive" property returns the result of the alarm statistics, stating how many active alarms were inactive on average.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.AverageActiveInactive
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Class

Description

The "Class" property returns the name of the alarm class.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.Class
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ClassSymbol

Description

The "ClassSymbol" property returns the icon of the alarm class.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.ClassSymbol
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ClearTime

Description

The "ClearTime" property returns the time of the alarm reset.

Type

DateTime

Access

Read-only

Syntax

```
AlarmBlockResult.ClearTime
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Duration

Description

The "Duration" property returns the duration of the alarm.

Type

Time

Access

Read-only

Syntax

```
AlarmBlockResult.Duration
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.EventText

Description

The "EventText" property returns a localized text describing an event to the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.EventText
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Frequency

Description

The "Frequency" property returns the result of the alarm statistics, stating how often the alarm occurred.

Type

UInt32

Access

Read-only

Syntax

```
AlarmBlockResult.Frequency
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.HostName

Description

The "HostName" property returns the name of the PC on which the alarm was triggered.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.HostName
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ID

Description

The "ID" property returns the alarm ID.

Type

UInt32

Access

Read-only

Syntax

```
AlarmBlockResult.ID
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.InfoText

Description

The "InfoText" property returns the text information.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.InfoText
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.LoopInAlarm

Description

The "LoopInAlarm" property returns the name of the function that navigates from the alarm display to its origin.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.LoopInAlarm
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ModificationTime

Description

The "ModificationTime" property returns the time stamp of the last modification to the alarm state.

Type

DateTime

Access

Read-only

Syntax

```
AlarmBlockResult.ModificationTime
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Name

Description

The "Name" property returns the name of the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.Name
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Origin

Description

The "Origin" property returns the origin of the alarm.

For example, the "Origin" property contains the system name, data source, or CPU ID. You can sort and filter alarms using the "Origin" context.

The "Origin" property can be configured and, together with the "Area" property, defines the source of an alarm. You can also use placeholders for context-sensitive format.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.Origin
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Priority

Description

The "Priority" property returns the relevance of the alarm or of the machine status.

Type

UInt8

Access

Read-only

Syntax

```
AlarmBlockResult.Priority
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue1

Description

The "ProcessValue1" property returns process value 1 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue1
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue2

Description

The "ProcessValue2" property returns process value 2 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue2
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue3

Description

The "ProcessValue3" property returns process value 3 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue3
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue4

Description

The "ProcessValue4" property returns process value 4 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue4
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue5

Description

The "ProcessValue5" property returns process value 5 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue5
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue6

Description

The "ProcessValue6" property returns process value 6 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue6
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue7

Description

The "ProcessValue7" property returns process value 7 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue7
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue8

Description

The "ProcessValue8" property returns process value 8 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue8
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue9

Description

The "ProcessValue9" property returns process value 9 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue9
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ProcessValue10

Description

The "ProcessValue10" property returns process value 10 of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ProcessValue10
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.RaiseTime

Description

The "RaiseTime" property returns the raise time of the alarm.

Type

DateTime

Access

Read-only

Syntax

```
AlarmBlockResult.RaiseTime
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ResetTime

Description

The "ResetTime" property returns the time of the alarm reset.

Type

DateTime

Access

Read-only

Syntax

```
AlarmBlockResult.ResetTime
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.SumActiveAcknowledged

Description

The "SumActiveAcknowledged" property returns the result of the alarm statistics, stating how many active alarms were acknowledged in total.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.SumActiveAcknowledged
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.SumActiveActive

Description

The "SumActiveActive" property returns the result of the alarm statistics, stating how many active alarms are active in total.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.SumActiveActive
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.SumActiveInactive

Description

The "SumActiveInactive" property returns the result of the alarm statistics, stating how many active alarms are inactive in total.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.SumActiveInactive
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.SuppressionState

Description

The "SuppressionState" property returns the visibility status of the alarm.

Value	SuppressionState	Description
0x0	Unsuppressed	Alarm is visible.
0x1	Suppressed	Alarm is configured as not visible.
0x3	Shelved	Alarm was hidden manually. The "Unshelve" and "Shelve" methods can be applied.

Type

UInt8

Access

Read-only

Syntax

```
AlarmBlockResult.SuppressionState
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.UserName

Description

The "UserName" property returns the name of the user who triggered the alarm.

Type

String

Access

Read-only

Syntax

```
AlarmBlockResult.UserName
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.Value

Description

The "Value" property returns the process value of the alarm.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.Value
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ValueLimit

Description

The "ValueLimit" property returns the limit of the alarm process value.

Type

Variant

Access

Read-only

Syntax

```
AlarmBlockResult.ValueLimit
```

See also

AlarmBlockResult (Page 1481)

AlarmBlockResult.ValueQuality

Description

The "ValueQuality" property returns the quality level of the process value of the alarm.

Type

UInt16

Access

Read-only

Syntax

```
AlarmBlockResult.ValueQuality
```

See also

AlarmBlockResult (Page 1481)

UI.DataSet

Description

The "DataSet" property returns the global data set ("DataSet" object) of the session.

Type

Object, HMIDataSet (Page 1510)

Access

Read-only

Syntax

```
[HMIRuntime.]UI.DataSet
```

See also

UI (Page 1395)

DataSet (Page 1510)

DataSet

Description

The "DataSet" object ("HMIDataSet" type) is a list of "DataSetElement" objects that enable data exchange in runtime.

A "DataSet" object is defined globally (on the "UI" object) or on a screen (the "Screen" object). You can access the data from any action.

You reference a "DataSet" object via the `UI.DataSet` property or via the respective screen with the `Screen.DataSet` property.

The "DataSet" object exists as long as the screen is displayed. The global object exists over the entire time period in which runtime runs.

Use

The "DataSet" object is a list which be counted and enumerated. You can access the "DataSet" list via the index or the element names.

Object type

HMIDataSet

Properties

The "DataSet" object has the following properties:

- **Count**
  Returns the number of elements of the "DataSet" list.

Methods

The "DataSet" object has the following methods:

- **Add()**
  Adds an element with value to the "DataSet" list.
- **Clear()**
  Removes all elements from the "DataSet" list.
- **Exists()**
  Checks whether an element is present in the "DataSet" list.
- **Item()**
  Returns the value of an element of the "DataSet" list.
- **Remove()**
  Removes an element by its name from the "DataSet" list.

DataSet.Count

Description

The "Count" property returns the number of elements in the "DataSet" list.

Type

UInt32

Access

Read-only

Syntax

```
DataSet.Count
```

See also

DataSet (Page 1510)

DataSet.Add()

Description

The "Add" method adds an element ("DataSetElement" object) to the "DataSet" list. The elements are referenced by name.

Syntax

```
DataSet.Add(name,value)
```

Parameters

name

Type: String

Name of the element that will be added to the list.

value

Type: Variant

Value of the element that will be added to the list.

Return value

--

See also

DataSet (Page 1510)

DataSet.Clear()

Description

The "Clear" method removes all elements ("DataSetElement" objects) from the "DataSet" list.

Syntax

```
DataSet.Clear()
```

Parameter

--

Return value

--

See also

DataSet (Page 1510)

DataSet.Remove() (Page 1514)

DataSet.Exists()

Description

The "Exists" method searches for an element ("DataSetElement" object) in the "DataSet" list.

Syntax

```
DataSet.Exists(name)
```

Parameters

name

Type: String

Name of the element that is being searched for.

Return value

Bool

See also

DataSet (Page 1510)

DataSet.Item()

Description

The "Item" method returns the value of a "DataSetElement" object of the "DataSet" list.

Syntax

```
DataSet[.Item](name)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "DataSet" object.

Parameter

`name`

Type: String

Element name or index number (1...n) of an object "DataSetElement" of the list

Return value

Variant

See also

DataSet (Page 1510)

DataSetElement (Page 1514)

DataSet.Remove()

Description

The "Remove" method removes an element ("DataSetElement" object) from the "DataSet" list by its name.

Syntax

```
DataSet.Remove(name)
```

Parameters

name

Type: String

Name of the element that will be removed.

Return value

--

See also

DataSet (Page 1510)

DataSet.Clear() (Page 1512)

DataSetElement

Description

The "DataSetElement" object represents a data set element in the "DataSet" list.

Object type

HMIDataSetElement

Properties

The "DataSetElement" object has the following properties:

- **Name**
  Returns the name of the element.
- **Value**
  Returns the value of the element.

Methods

--

DataSetElement.Name

Description

The "Name" property returns the name of the element ("DataSetElement" object).

Type

String

Access

Read-only

Syntax

```
DataSetElement.Name
```

See also

DataSetElement (Page 1514)

DataSetElement.Value

Description

The "Name" property returns the value of the element ("DataSetElement" object).

Type

Variant

Access

Read-only

Syntax

```
DataSetElement.Value
```

See also

DataSetElement (Page 1514)

UI.PopupScreenWindows

Description

The "Windows" property returns a list of all popup screen windows ("HMIPopupScreenWindows" objects).

Type

Object, HMIPopupScreenWindows (Page 1516)

Access

Read-only

Syntax

```
[HMIRuntime.]UI.PopupScreenWindows
```

See also

UI (Page 1395)

PopupScreenWindows (Page 1516)

PopupScreenWindows

Description

The "PopupScreenWindows" object is a list of all popup screen windows ("PopupScreenWindow" objects).

Use

The "PopupScreenWindows" object is a list which can be counted and enumerated. You can access the "PopupScreenWindows" list using the index or the tag names.

Object type

HMIPopupScreenWindows

Properties

The "PopupScreenWindows" object has the following properties:

- **Count**
  Returns the number of popup screen windows in the "PopupScreenWindows" list.

Methods

The "PopupScreenWindows" object has the following methods:

- **Item()**
  Returns a popup screen window of the "PopupScreenWindows" list.

See also

UI.PopupScreenWindows (Page 1516)

PopupScreenWindows.Count

Description

The "Count" property returns the number of popup screen windows in the "PopupScreenWindows" list.

Type

UInt32

Access

Read-only

Syntax

```
PopupScreenWindows.Count
```

See also

PopupScreenWindows (Page 1516)

PopupScreenWindows.Item()

Description

The "Item" method returns a popup screen window of the "PopupScreenWindows" list.

Syntax

```
PopupScreenWindows[.Item](WindowName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "PopupScreenWindows" object.

Parameter

`WindowName`

Type: String, HMIPopupScreenWindow

Name of the popup screen window

Return value

Object, HMIPopupScreenWindow (Page 4288)

See also

PopupScreenWindows (Page 1516)

PopupScreenWindow (Page 4288)

PopupScreenWindow

Description

PopupScreenWindow (Page 4288)

UI.RootWindow

Description

The "RootWindow" property returns the top level (Top Level Screen Window) screen window of the screen where the script is executed.

Type

Object, HmiTopLevelScreenWindow (Page 1522)

Access

Read-only

Syntax

```
[HMIRuntime.]UI.RootWindow
```

See also

UI (Page 1395)

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow

Description

TopLevelScreenWindow (Page 1522)

UI.Style

Description

The "Style" property specifies the style of the display and operating objects.

Type

String, HmiStyle

Access

Read-write

Syntax

```
[HMIRuntime.]UI.Style
```

Example

Switch all objects in runtime to the dark style and then output the style via the debug output:

Copy code

```
HMIRuntime.UI.Style = "FlatStyle_Dark";
HMIRuntime.Trace("Switched style to: " + HMIRuntime.UI.Style);
```

See also

UI (Page 1395)

UI.Windows

Description

The "Windows" property returns a list of all screen windows at the top level (Top Level Screen Window).

Type

Object, HMITopLevelWindows (Page 1520)

Access

Read-only

Syntax

```
[HMIRuntime.]UI.Windows
```

See also

UI (Page 1395)

TopLevelWindows (Page 1520)

TopLevelWindows

Description

The "TopLevelWindows" object is a list of "TopLevelScreenWindow" objects that provides access to the top-level screen windows in runtime.

You reference a "TopLevelWindows" object via the `UI.Windows` property.

Use

The "TopLevelWindows" object is a list and can be counted and enumerated. You can access the "TopLevelWindows" list using the index or the tag name.

Object type

HMITopLevelWindows

Properties

The "TopLevelWindows" object has the following properties:

- **Count**
  Returns the number of screen windows at the highest level of the "TopLevelWindows" list.

Methods

The "TopLevelWindows" object has the following methods:

- **Item()**
  Returns a screen window at the highest level of the "TopLevelWindows" list.

TopLevelWindows.Count

Description

The "Count" property returns the number of screen windows at the highest level of the "TopLevelWindows" list.

Type

UInt32

Access

Read-only

Syntax

```
TopLevelWindows.Count
```

See also

TopLevelWindows (Page 1520)

TopLevelWindows.Item()

Description

The "Item" method returns a screen window at the highest level of the "TopLevelWindows" list.

Syntax

```
TopLevelWindows[.Item](WindowName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "TopLevelWindows" object.

Parameter

`WindowName`

Type: String, HmiTopLevelScreenWindow

Name of the screen window at the highest level

Return value

Object, HmiTopLevelScreenWindow (Page 1522)

See also

TopLevelWindows (Page 1520)

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow

Description

The "TopLevelScreenWindows" object represents a screen window at the highest hierarchy level.

Object type

HmiTopLevelScreenWindow

Properties

The "TopLevelScreenWindow" object has the following properties:

- **Adaption**
  Specifies how the window size adapts.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.

- **CurrentScreen**
  Returns the screen of the current screen window.
- **CurrentZoomFactor**
  Specifies the zoom factor which is applied to the displayed screen.
- **Enabled**
  Specifies whether the specified object can be operated in runtime.
- **Height**
  Specifies the height.
- **HorizontalScrollBarPosition**
  Specifies the horizontal alignment for the scroll bar.
- **HorizontalScrollBarVisibility**
  Specifies the setting for the horizontal scroll bar of the window.
- **Icon**
  Specifies the icon.
- **InteractiveZooming**
  Specifies whether zooming is supported.
- **Layer**
  Returns the layer of the screen that contains the screen window.
- **Left**
  Specifies the value of the X coordinate.
- **Monitor**
  Returns the monitor on which the screen window is displayed.
- **Name**
  Returns the name of the screen window.
- **Parent**
  Returns the parent screen object (Parent container).
- **Path**
  Returns the absolute object path of the screen window.
- **RenderingTemplate**
  Returns the name of the template from which the object was created.
- **Screen**
  Specifies the name of the screen contained in the referenced screen window.
- **ScreenName**
  Returns the screen name.
- **ScreenNumber**
  Returns the screen number.
- **StartupPosition**
  Specifies the position of the screen window at runtime start.
- **StyleItemClass**
  Returns the style that is applied to the screen window.
- **System**
  Specifies the server prefix.

- **TabIndex**
  Returns the position of the screen window in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **VerticalScrollBarPosition**
  Specifies the vertical alignment for the scroll bar.
- **VerticalScrollBarVisibility**
  Specifies the setting for the vertical scroll bar of the window.
- **Visible**
  Specifies whether the screen window is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the screen window configuration.

Methods

The "TopLevelScreenWindow" object has the following methods:

- **Close()**
  Closes the screen window.

TopLevelScreenWindow.Adaption

Description

The "Adaption" property specifies how the window size adapts.

Type

Int32, HmiScreenWindowAdaption

Specifies how the window size adapts:

- None (0): No adaptation
- WindowToScreen (1): Window size corresponds to screen size
- ScreenToWindow (2): Screen is scaled to window size.

Access

Read-write

Syntax

```
TopLevelScreenWindow.Adaption
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
TopLevelScreenWindow.Caption
```

See also

TopLevelScreenWindow (Page 1522)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

TopLevelScreenWindow.Caption (Page 1525)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 1525)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 1525)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 1525)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 1525)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 1525)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 1525)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

TopLevelScreenWindow.Caption (Page 1525)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

TopLevelScreenWindow.Caption (Page 1525)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

TopLevelScreenWindow.Caption (Page 1525)

TopLevelScreenWindow.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
TopLevelScreenWindow.CaptionColor
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.CurrentScreen

Description

The "CurrentScreen" property returns the screen of the current screen window.

Type

Object, HmiScreen (Page 1397)

Access

Read-only

Syntax

```
TopLevelScreenWindow.CurrentScreen
```

See also

TopLevelScreenWindow (Page 1522)

Screen (Page 1397)

Screen

Description

Screen (Page 1397)

TopLevelScreenWindow.CurrentZoomFactor

Description

The "CurrentZoomFactor" property specifies the zoom factor of the screen window. The zoom factor may differ from the containing screen. The value 1.0 corresponds to a zoom factor of 100%.

Type

Float

Access

Read-write

Syntax

```
TopLevelScreenWindow.CurrentZoomFactor
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Enabled

Description

The "Enabled" property specifies whether the screen window can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TopLevelScreenWindow.Enabled
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Height

Description

The "Height" property specifies the height of the screen window in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TopLevelScreenWindow.Height
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.HorizontalScrollBarPosition

Description

The "HorizontalScrollBarPosition" property specifies the horizontal alignment for the scroll bar.

Type

Int32

Access

Read-write

Syntax

```
TopLevelScreenWindow.HorizontalScrollBarPosition
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.HorizontalScrollBarVisibility (Page 1534)

TopLevelScreenWindow.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
TopLevelScreenWindow.HorizontalScrollBarVisibility
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.HorizontalScrollBarPosition (Page 1533)

TopLevelScreenWindow.Icon

Description

The "Icon" property specifies the icon of the screen window.

Type

String

Access

Read-write

Syntax

```
TopLevelScreenWindow.Icon
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.InteractiveZooming

Description

The "InteractiveZooming" property specifies whether zooming is supported.

Type

Bool

Access

Read-write

Syntax

```
TopLevelScreenWindow.InteractiveZooming
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Layer

Description

The "Layer" property returns the layer of the screen that contains the screen window.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
TopLevelScreenWindow.Layer
```

See also

TopLevelScreenWindow (Page 1522)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Layer.MinimumZoom (Page 1536)

TopLevelScreenWindow.Layer (Page 1535)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Layer.MaximumZoom (Page 1536)

TopLevelScreenWindow.Layer (Page 1535)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

TopLevelScreenWindow.Layer (Page 1535)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

TopLevelScreenWindow.Layer (Page 1535)

TopLevelScreenWindow.Left

Description

The "Left" property specifies the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TopLevelScreenWindow.Left
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Monitor

Description

The "Monitor" property returns the monitor on which the window is displayed.

Type

UInt8

Access

Read-only

Syntax

```
TopLevelScreenWindow.Monitor
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Name

Description

The "Name" property returns the name of the screen window.

Type

String

Access

Read-only

Syntax

```
TopLevelScreenWindow.Name
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Parent

Description

The "Parent" property returns the parent screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
TopLevelScreenWindow.Parent
```

See also

TopLevelScreenWindow (Page 1522)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

TopLevelScreenWindow.Path

Description

The "Path" property returns the absolute object path of a screen window in runtime starting from the screen window on the highest level.

Note

For the syntax of an object path, see the "FindItem" method (Page 1549).

Type

String

Access

Read-only

Syntax

```
TopLevelScreenWindow.Path
```

See also

UI.FindItem() (Page 1549)

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the object has been created.

Type

String

Access

Read-only

Syntax

```
TopLevelScreenWindow.RenderingTemplate
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Screen

Description

The "Screen" property specifies the name of the screen ("HMIScreen" type) that is contained in the screen window. Loads a new screen via its name into the screen window.

The "Screen" property returns a different value than the "CurrentScreen" when the referenced screen is not yet loaded completely or does not exist.

Type

String, HmiStoredScreen

Access

Read-write

Syntax

```
TopLevelScreenWindow.Screen
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.ScreenName

Description

The "ScreenName" property returns the screen name.

Type

String

Access

Read-only

Syntax

```
TopLevelScreenWindow.ScreenName
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.ScreenNumber

Description

The "ScreenNumber" property returns the screen number.

Type

UInt16

Access

Read-only

Syntax

```
TopLevelScreenWindow.ScreenNumber
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.StartupPosition

Description

The "StartupPosition" property sets the position of the screen window at runtime start.

Type

Int32, HmiWindowStartupPosition

Specifies the position of the screen window:

- None (0): Relative placement on the configured monitor via "Left" and "Top".
- CenteredMonitor (1): Centered on the configured monitor.
- Maximized (2): Maximized on the configured monitor.
- CenteredOwner (3): Centered on the displayed screen.

Access

Read-write

Syntax

```
TopLevelScreenWindow.StartupPosition
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.StyleItemClass

Description

The "StyleItemClass" property returns the style that is applied to the screen window.

Type

String

Access

Read-only

Syntax

```
TopLevelScreenWindow.StyleItemClass
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.System

Description

The "System" property specifies the server prefix.

Type

String, HmiSystem

Access

Read-write

Syntax

```
TopLevelScreenWindow.System
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.TabIndex

Description

The "TabIndex" returns the position of the screen window in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
TopLevelScreenWindow.TabIndex
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Top

Description

The "Top" property specifies the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TopLevelScreenWindow.Top
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.VerticalScrollBarPosition

Description

The "VerticalScrollBarPosition" property specifies the vertical position for the scroll bar.

Type

Int32

Access

Read-write

Syntax

```
TopLevelScreenWindow.VerticalScrollBarPosition
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.VerticalScrollBarVisibility (Page 1546)

TopLevelScreenWindow.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the window.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
TopLevelScreenWindow.VerticalScrollBarVisibility
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.VerticalScrollBarPosition (Page 1545)

TopLevelScreenWindow.Visible

Description

The "Visible" property specifies whether the screen window is visible.

Type

Bool

Access

Read-write

Syntax

```
TopLevelScreenWindow.Visible
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Width

Description

The "Width" property specifies the width of the screen window in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TopLevelScreenWindow.Width
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.WindowFlags

Description

The "WindowFlags" property specifies the screen window configuration.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
TopLevelScreenWindow.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

TopLevelScreenWindow (Page 1522)

TopLevelScreenWindow.Close()

Description

The "Close" method closes the screen window.

Syntax

```
TopLevelScreenWindow.Close()
```

Parameters

--

Return value

Bool

See also

TopLevelScreenWindow (Page 1522)

UI.FindItem()

Description

The "FindItem" method returns screen windows or screen objects through their object path.

Syntax

```
[HMIRuntime.]UI.FindItem(ScreenItemPath);
```

Parameter

`ScreenItemPath`

Type: String

Object path of the searched screen window or screen object.

Note

The "UI.FindItem" method has a global search context and requires absolute object paths. The "Screen.FindItem" method has the current screen as the search context and can also use relative object paths.

Formulation of the object path

The syntax of the object path orients itself to the notation of the file system paths. The object path consists of the names of the screen windows (Screen Windows) and screen objects (Screen Items). The names are connected via a slash ("/") according to the hierarchical positioning. Screens (Screens) and their names are not used in the formulation.

Relative and absolute object paths are distinguished by the prefix of the object path. The following prefixes can be used:

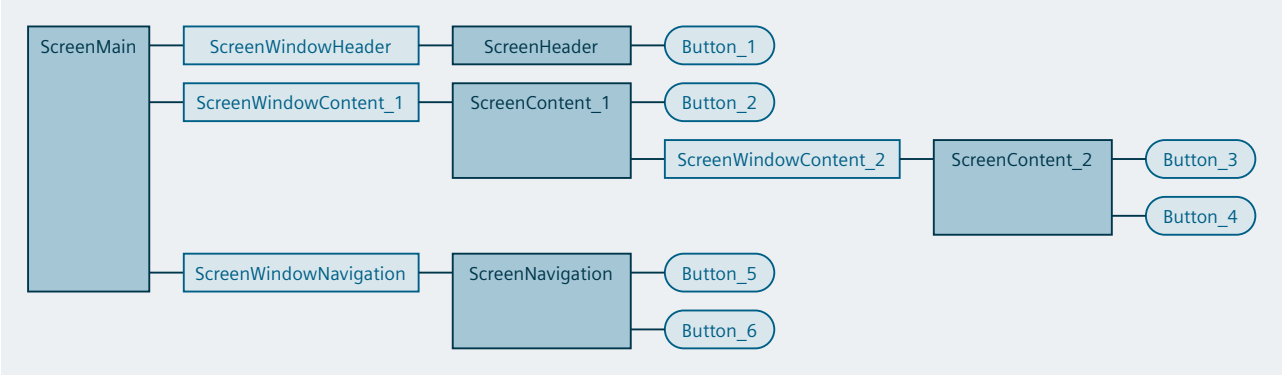
- Relative object path
  - "..": References the parent screen window (Parent) in the context of the current screen window.
  - ".": References the own screen window (Self).
  - "": A screen object of the current screen window is referenced without prefix.
- Absolute object path
  - "/": References a screen window on the highest level, whose name must follow.
  - "~": References the screen window on the highest level in the own screen hierarchy.

Additional rules for formulating an object path:

- The string ".." may be used several times in the object path, but only together at the beginning of the object path, for example, "../../Window5".
- If the object path does not end with a screen object name, a screen window is referenced.
- A search is performed for screen objects of the object path in the screens of the referenced screen window. It is not permitted to specify a screen name.

Examples of object paths

The following window / screen object hierarchy is adopted for the following examples:



The following objects paths for addressing the object result from this:

- Relative addressing:
  - "Button_2" changes the label of "Button_1":
    ```
    // Navigate one level up and find "Button_1" inside the
    "ScreenWindowHeader"
    Screen.FindItem("../ScreenWindowHeader/Button_1").Text =
    "Changed by Button_2"
    ```
  - "Button_3" changes the label of "Button_5":
    ```
    // Navigate two levels up and find "Button 2" inside the
    "ScreenWindowNavigation"
    Screen.FindItem("../../ScreenWindowNavigation/Button_5").Text =
    "Changed by Button_3"
    ```
  - "Button_3" changes the label of "Button_4":
    ```
    // Find "Button_4" in same screen ("ScreenContent_2")
    Screen.FindItem("Button_4").Text = "Changed by Button_3"
    Screen.FindItem("./Button_4").Text = "Changed by Button_3"
    ```
- Absolute addressing:
  - "Button_4" changes the label of "Button_6":
    ```
    // Navigate to the root screen and find "Button_6" inside the
    "ScreenWindowNavigation"
    Screen.FindItem("~/ScreenWindowNavigation/Button_6").Text =
    "Changed by Button_4"
    ```

Return value

Object, HmiScreenObjectBase (Page 1571)

See also

UI (Page 1395)

Screen Items (Page 1571)

UI.GetClientInfo()

Description

The "GetClientInfo" method returns information about the resolution of the screen.

Syntax

```
[HMIRuntime.]UI.GetClientInfo(infoType);
```

Parameters

`infoType`

Type: Int32, HmiClientInfoType

Specifies which information is requested:

- None (0): Does not return any information.
- PrimaryDisplayWidth (1): Returns the width of the screen in DIU (Device Independent Unit).
- PrimaryDisplayHeight (2): Returns the height of the screen in DIU (Device Independent Unit).
- PrimaryDisplayPixelRatio (3): Returns the ratio of the screen pixels in DIU (Device Independent Unit).

Return value

Variant

Example

Specify the screen window on the highest level depending on the resolution:

Copy code

```
let displayWidth = UI.GetClientInfo(UI.Enums.HmiClientInfoType.PrimaryDisplayWidth);
let displayHeight = UI.GetClientInfo(UI.Enums.HmiClientInfoType.PrimaryDisplayHeight);

if (displayHeight > displayWidth) {
   UI.RootWindow.Screen = "StartScreenPortrait";
} else {
   UI.RootWindow.Screen = "StartScreenLandscape";
}
```

See also

UI (Page 1395)

UI.OpenFaceplateInPopup()

Description

The "OpenFaceplateInPopup" method opens a faceplate from a screen in a popup window.

Syntax

```
[HMIRuntime.]UI.OpenFaceplateInPopup(faceplateType,title,interface[,
parentScreen][,invisible]);
```

Parameters

`faceplateType`

Type: String, HmiFaceplateType

Name of the faceplate type

Note

The version of the faceplate type preset as "Default" is automatically displayed in runtime.

`title`

Type: String

Title of the popup window

`interface`

Type: Object

Interface data of the faceplate in literal notation, e.g. `{Interface_Tag_1:{Tag:"HMI_Tag_1"}, Color_Property_1:0xff00ff00, ResourceList_Property_1:"@Default.Text_list_1"}`. The specification of the interface data by entering a graphics list is not supported.

`parentScreen`

Optional, type: Object, HmiScreen

Parent screen

`invisible`

Optional, type: Bool

Causes the faceplate to be configured so that it is invisible to the operator.

To then display the faceplate, set the property `visible=true`.

Return value

Object, HmiPopupScreenWindow (Page 4288)

See also

UI (Page 1395)

PopupScreenWindow (Page 4288)

ProDiag

Description

The "ProDiag" object

Object type

HMIUIProDiag

Properties

--

Methods

The "ProDiag" object has the following methods:

- **OpenTIAPortalFromAlarm()**
  Opens a TIA Portal project and jumps to the "GRAPH" details block.
- **OpenTIAPortalGRAPHDetails()**
  Opens a TIA Portal project and jumps to the "GRAPH" details block.
- **OpenTIAPortalProDiagDetailsByAssignment()**
  Opens a TIA Portal project and jumps to the "GRAPH" details block.
- **OpenTIAPortalProDiagDetailsByCall()**
  Opens a TIA Portal project and jumps to the "GRAPH" details block.
- **OpenTIAPortalProject()**
  Opens a TIA Portal project.

See also

UI (Page 1395)

ProDiag.OpenTIAPortalFromAlarm()

Description

The "OpenTIAPortalFromAlarm" method opens a TIA Portal project and jumps to the "GRAPH" details block.

Syntax

```
[HMIRuntime.]UI.ProDiag.OpenTIAPortalFromAlarm(DontUseModifiedProjec
t, ReadOnly, Online, AlarmView, pathToProject);
```

Parameters

`DontUseModifiedProject`

Type: Bool

If a changed project is already open, the project will not be opened.

ReadOnly

Type: Bool

Opens the project write-protected.

Online

Type: Bool

Goes into online mode after opening the block.

AlarmView

Type: String

Path of the alarm control

pathToProject

Type: String

Path of the project

Return value

ErrorCode

See also

ProDiag (Page 1553)

ProDiag.OpenTIAPortalGRAPHDetails()

Description

The "OpenTIAPortalGRAPHDetails" method opens a TIA Portal project and jumps to the "GRAPH" details block.

Syntax

```
[HMIRuntime.]UI.ProDiag.OpenTIAPortalGRAPHDetails(DontUseModifiedPro
ject, ReadOnly, Online, PlcName, Block, pathToProject);
```

Parameters

DontUseModifiedProject

Type: Bool

If a changed project is already open, the project will not be opened.

ReadOnly

Type: Bool

Opens the project write-protected.

`Online`

Type: Bool

Goes into online mode after opening the block.

`PlcName`

Type: String

Name of the PLC

`Block`

Type: String

Name of the block

`pathToProject`

Type: String

Path of the project

Return value

ErrorCode

See also

ProDiag (Page 1553)

ProDiag.OpenTIAPortalProDiagDetailsByAssignment()

Description

The "OpenTIAPortalProDiagDetailsByAssignment" method opens a TIA Portal project and jumps to the "GRAPH" details block.

Syntax

```
[HMIRuntime.]UI.ProDiag.OpenTIAPortalProDiagDetailsByAssignment(Dont
UseModifiedProject, ReadOnly, Online, PlcName, Block, Operand
pathToProject);
```

Parameters

`DontUseModifiedProject`

Type: Bool

If a changed project is already open, the project will not be opened.

`ReadOnly`

Type: Bool

Opens the project write-protected.

`Online`

Type:Bool

Goes into online mode after opening the block.

`PlcName`

Type: String

Name of the PLC

`Block`

Type: String

Name of the block

`Operand`

Type: String

Operand of the access point

`pathToProject`

Type: String

Path of the project

Return value

ErrorCode

See also

ProDiag (Page 1553)

ProDiag.OpenTIAPortalProDiagDetailsByCall()

Description

The "OpenTIAPortalProDiagDetailsByCall" method opens a TIA Portal project and jumps to the "GRAPH" details block.

Syntax

```
[HMIRuntime.]UI.ProDiag.OpenTIAPortalProDiagDetailsByCall(DontUseMod
ifiedProject, ReadOnly, Online, PlcName, ContainingBlock,
CalledBlock, Pin, pathToProject);
```

Parameters

`DontUseModifiedProject`

Type: Bool

If a changed project is already open, the project will not be opened.

`ReadOnly`

Type: Bool

Opens the project write-protected.

`Online`

Type: Bool

Goes into online mode after opening the block.

`PlcName`

Type: String

Name of the PLC

`ContainingBlock`

Type: String

Name of the contained block

`CalledBlock`

Type: String

Name of the called block

`Pin`

Type: String

Access point PIN

`pathToProject`

Type: String

Path of the project

Return value

ErrorCode

See also

ProDiag (Page 1553)

ProDiag.OpenTIAPortalProject()

Description

The "OpenTIAPortalProject" method opens a TIA Portal project.

Syntax

```
[HMIRuntime.]UI.ProDiag.OpenTIAPortalProject(DontUseModifiedProject,
 ReadOnly, pathToProject);
```

Parameters

DontUseModifiedProject

Type: Bool

If a changed project is already open, the project will not be opened.

ReadOnly

Type: Bool

Opens the project write-protected.

pathToProject

Type: String

Path of the project

Return value

ErrorCode

See also

ProDiag (Page 1553)

SysFct

Description

The "SysFct" object enables access to the system functions of the "ProDiag" object.

Object type

HMIUIProDiagSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **IsJumpableAlarm()**
  Checks whether the alarm selected in the alarm control is a ProDiag alarm.
- **Next()**
  Executes the "Next" command in the PLC code viewer.
- **OpenPlcCodeViewByFCCall()**
  Represents the logic of a network input of a standard block in the PLC code viewer display taking the UDT instance into account.
- **OpenPlcCodeViewFromAlarm()**
  Opens the corresponding block in the PLC code view according to the selection in the alarm control.
- **OpenProDiagDetailsByAssignment()**
  Represents an assignment of an operand and its logic in the PLC code viewer display.
- **OpenProDiagDetailsByCall()**
  Represents the logic of a network input of a standard block in the PLC code viewer display.
- **OpenViewerGraphByBlock()**
  Jumps to the PLC code viewer and opens an S7 GRAPH step.
- **OpenViewerGraphFromOverview()**
  Jumps from a graph overview control to the PLC code viewer.
- **Previous()**
  Executes the "Previous" command in the PLC code viewer.
- **ResetToConfiguration()**
  Executes the "ResetToConfiguration" command in the PLC code viewer.
- **ToggleCriteriaAnalysis()**
  Executes the "ToggleCriteriaAnalysis" command in the PLC code viewer.
- **ToggleGRAPHViewerMode()**
  Executes the "ToggleGRAPHViewerMode" command in the PLC code viewer.
- **ToggleNetworkDisplay()**
  Executes the "ToggleNetworkDisplay" command in the PLC code viewer.
- **ZoomIn()**
  Executes the "ZoomIn" command in the PLC code viewer.
- **ZoomOut()**
  Executes the "ZoomOut" command in the PLC code viewer.

See also

ProDiag (Page 1553)

SysFct.IsJumpableAlarm()

Description

The "IsJumpableAlarm" method checks whether the alarm selected in the alarm control is a ProDiag alarm. If yes, the specified screen entry is enabled. Otherwise it is disabled.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.IsJumpableAlarm(AlarmView,
screenItemPath);
```

Parameters

AlarmView

Type: String, HmiAlarmControl

Path of the alarm control with selected alarm

screenItemPath

Type: String, HmiScreenItemBase

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.Next()

Description

The "Next" method executes the "Next" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.Next(screenItemPathCodeViewer);
```

Parameters

screenItemPathCodeViewer

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.OpenPlcCodeViewByFCCall()

Description

The "OpenPlcCodeViewByFCCall" represents the logic of a network input of a standard block in the PLC code viewer display, taking the UDT instance into account.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.OpenPlcCodeViewByFCCall(pinSubstringS
earch, plcName, containingBlock, calledBlock, pin, UdtInstance,
screenItemPath);
```

Parameters

pinSubstringSearch

Type: Bool

Specifies whether the pin name starts with the transferred pin parameter.

- True: Pin name starts with the transferred pin parameter
- False: Pin name must be the same as the pin parameter

plcName

Type: String

Name of the PLC

containingBlock

Type: String

Name of the contained block

calledBlock

Type: String

Name of the called block

pin

Type: String

Name of the input pin of the "CalledBlock".

UdtInstance

Type: String

UDT instance that is used to limit the display of FCs called multiple times.

screenItemPath

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.OpenPlcCodeViewFromAlarm()

Description

The "OpenPlcCodeViewFromAlarm" method opens the corresponding block in the PLC code view according to the selection in the alarm control.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.OpenPlcCodeViewFromAlarm(AlarmView,
PlcCodeView);
```

Parameters

AlarmView

Type: String

Path of the alarm control

PlcCodeView

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path to the PLC code view.

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.OpenProDiagDetailsByAssignment()

Description

The "OpenProDiagDetailsByAssignment" method represents an operand assignment and its logic in the PLC code viewer display.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.OpenProDiagDetailsByAssignment(plcNam
e, containingBlock, operand, screenItemPath);
```

Parameters

plcName

Type: String

Name of the PLC

containingBlock

Type: String

Name of the contained block

operand

Type: String

Operand of the access point

screenItemPath

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.OpenProDiagDetailsByCall()

Description

The "OpenProDiagDetailsByCall" represents the logic of a network input of a standard block in the PLC code viewer display.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.OpenProDiagDetailsByCall(pinSubstring
Search, plcName, containingBlock, calledBlock, pin, screenItemPath);
```

Parameters

pinSubstringSearch

Type: Bool

Specifies whether the pin name starts with the transferred pin parameter.

- True: Pin name starts with the transferred pin parameter
- False: Pin name must be the same as the pin parameter

plcName

Type: String

Name of the PLC

containingBlock

Type: String

Name of the contained block

calledBlock

Type: String

Name of the called block

pin

Type: String

Name of the input pin of the "CalledBlock".

screenItemPath

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.OpenViewerGraphByBlock()

Description

The "OpenViewerGraphByBlock" method jumps to the PLC code viewer and opens an S7 GRAPH step.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.OpenViewerGraphByBlock(plcName,
graphBlockName, stepNumber, screenItemPath);
```

Parameters

`plcName`

Type: String

Name of the PLC

`graphBlockName`

Type: String

Name of the GRAPH block

`stepNumber`

Type: UInt32

Step number

`screenItemPath`

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.OpenViewerGraphFromOverview()

Description

The "OpenViewerGraphFromOverview" method jumps from a graph overview control to the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.OpenViewerGraphFromOverview(screenIte
mPathOverviewControl, screenItemPathCodeViewer);
```

Parameters

screenItemPathOverviewControl

Type: String, HmiProcessDiagnosisGraphOverviewControl

Path of the graph overview control

screenItemPathCodeViewer

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.Previous()

Description

The "Previous" method executes the "Previous" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.Previous(blub);
```

Parameters

screenItemPathCodeViewer

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.ResetToConfiguration()

Description

The "ResetToConfiguration" method executes the "ResetToConfiguration" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.ResetToConfiguration(screenItemPathCo
deViewer);
```

Parameters

`screenItemPathCodeViewer`

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.ToggleCriteriaAnalysis()

Description

The "ToggleCriteriaAnalysis" method executes the "ToggleCriteriaAnalysis" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.ToggleCriteriaAnalysis(screenItemPath
CodeViewer);
```

Parameters

`screenItemPathCodeViewer`

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.ToggleGRAPHViewerMode()

Description

The "ToggleGRAPHViewerMode" method executes the "ToggleGRAPHViewerMode" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.ToggleGRAPHViewerMode(screenItemPathC
odeViewer);
```

Parameters

`screenItemPathCodeViewer`

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.ToggleNetworkDisplay()

Description

The "ToggleNetworkDisplay" method executes the "ToggleNetworkDisplay" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.ToggleNetworkDisplay(screenItemPathCo
deViewer);
```

Parameters

screenItemPathCodeViewer

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.ZoomIn()

Description

The "ZoomIn" method executes the "ZoomIn" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.ZoomIn(screenItemPathCodeViewer);
```

Parameters

screenItemPathCodeViewer

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

SysFct.ZoomOut()

Description

The "ZoomOut" method executes the "ZoomOut" command in the PLC code viewer.

Syntax

```
[HMIRuntime.]UI.ProDiag.SysFct.ZoomOut(screenItemPathCodeViewer);
```

Parameters

`screenItemPathCodeViewer`

Type: String, HmiProcessDiagnosisPlcCodeViewerControl

Path of the code viewer display

Return value

ErrorCode

See also

SysFct (Page 1559)

Screen Items

AlarmControl

Description

The "AlarmControl" object represents an alarm control. It displays current or logged alarms of a plant in list form.

Object type

HMIAlarmControl

Properties

The "AlarmControl" object has the following properties:

- **AcknowledgmentFlashingRate**
  Specifies the flashing frequency for alarms that must be acknowledged.
- **ActiveAlarmsViewSetup**
  Specifies which predefined alarm filter is applied to pending alarms.
- **AlarmDefinitionViewSetup**
  Specifies which predefined alarm filter is applied for user-defined alarm definitions.
- **AlarmSourceType**
  Specifies the alarm source of the alarm control.
- **AlarmStatisticsSettings**
  Returns the "AlarmStatisticsSettings" object. Assigns parameters for the statistical calculations for logged alarms.
- **AlarmStatisticsView**
  Sets the "AlarmStatisticsView" object.
  Sets an alarm window with columns and alarm rows for statistical calculations on logged alarms.
- **AlarmView**
  Specifies the "AlarmView" object.
- **AlwaysShowRecent**
  Specifies whether the most recent alarm is displayed at the beginning or end of the list, depending on the sorting.
- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the color of the title bar.
- **CurrentQuality**
  Returns the current worst quality code of all tags which influence the alarm control.
- **DefaultSortDirection**
  Specifies the sorting order of the time column if no other sorting is active.
- **Enabled**
  Specifies whether the alarm control can be operated in runtime.
- **Filter**
  Specifies a string for filtering alarms.
- **Height**
  Specifies the height of the alarm control.
- **Icon**
  Specifies the icon of the alarm control.
- **Layer**
  Returns the layer of the screen where the alarm control is located.

- **Left**
  Specifies the value of the X coordinate of the alarm control.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the alarm control.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the object was created.
- **ResetFlashingRate**
  Sets the flashing frequency for alarms that need to be reset.
- **ShowFocusVisual**
  Specifies whether the alarm control is highlighted when in focus.
- **StatusBar**
  Sets the information bar of the alarm control.
- **StyleItemClass**
  Returns the style which is applied to the alarm control.
- **SuppressFlashing**
  Specifies whether flashing is suppressed.
- **Systems**
  Specifies the name of the runtime system for the grouping of active alarms.
- **TabIndex**
  Returns the position of the alarm control in the tab sequence.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Sets the toolbar of the alarm control.
- **Top**
  Specifies the value of the Y coordinate of the alarm control.
- **UseAlarmColors**
  Specifies whether the configured color of the alarm is used.
- **Visible**
  Specifies whether the alarm control is visible.
- **Width**
  Specifies the width of the alarm control.
- **WindowFlags**
  Specifies the window configuration of the alarm control.

Methods

The "AlarmControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the alarm control.
- **FireCommand()**
  Executes the command of an element of the toolbar or information bar of the alarm control.
- **GetSelectedAlarmData()**
  Returns all the data of the selected alarm of the alarm control.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "AlarmControl" object has the following events:

- **OnActivated()**
  Occurs when an alarm control receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the alarm control.
- **OnDeactivated()**
  Occurs when an alarm control loses focus.
- **OnInitialized()**
  Occurs when an alarm control has been successfully initialized and the data connection to the PLC has been established.
- **OnSelectionChanged()**
  Occurs when the selection in the alarm control changes.

AlarmControl.AcknowledgmentFlashingRate

Description

The "AcknowledgmentFlashingRate" property specifies the flashing frequency for alarms that must be acknowledged.

Type

Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Access

Read-write

Syntax

```
AlarmControl.AcknowledgmentFlashingRate
```

See also

AlarmControl (Page 1571)

AlarmControl.ActiveAlarmsViewSetup

Description

The "ActiveAlarmsViewSetup" property specifies which predefined alarm filter is applied to current alarms ("ActiveAlarms" alarm source).

Type

Int32, HmiVisibleAlarms

Specifies the displayed alarms:

- None (0): Show no alarms.
- UnSuppressed (1): Show all alarms that are not suppressed.
- Disabled (2): Show only locked alarms (only if alarm source "AlarmDefinition").
- SuppressedByDesign (4): Show only alarms suppressed by design.
- Shelved (8): Show only shelved alarms.

Access

Read-write

Syntax

```
AlarmControl.ActiveAlarmsViewSetup
```

See also

AlarmControl (Page 1571)

AlarmControl.AlarmDefinitionViewSetup

Description

The "AlarmDefinitionViewSetup" property specifies which predefined alarm filter is applied to displayed alarms from a configured alarm filter configuration (alarm source "AlarmDefinition").

Type

Int32, HmiVisibleAlarms

Specifies the displayed alarms:

- None (0): Show no alarms.
- UnSuppressed (1): Show all alarms that are not suppressed.
- Disabled (2): Show only locked alarms (only if alarm source "AlarmDefinition").
- SuppressedByDesign (4): Show only alarms suppressed by design.
- Shelved (8): Show only shelved alarms.

Access

Read-write

Syntax

```
AlarmControl.AlarmDefinitionViewSetup
```

See also

AlarmControl (Page 1571)

AlarmControl.AlarmSourceType

Description

The "AlarmSourceType" property specifies the alarm source of the alarm control.

Type

Int32, HmiAlarmSourceType

Specifies the alarm source:

- NotConfigured (0): Not defined
- ActiveAlarms (1): Pending alarms
- LoggedAlarms (2): Logged alarms

- LoggedAlarmsUpdated (3): Logged alarms with updates
- AlarmDefintion (4): Alarms from configured alarm filter configuration
- AlarmStatistics (5): Alarm statistics

Access

Read-write

Syntax

```
AlarmControl.AlarmSourceType
```

See also

AlarmControl (Page 1571)

AlarmControl.AlarmStatisticsSettings

Description

The "AlarmStatisticsSettings" property specifies time settings for statistical calculations on logged alarms.

Type

Object, HmiAlarmStatisticsSettingsPart

Access

Read-write

Syntax

```
AlarmControl.AlarmStatisticsSettings
```

See also

AlarmControl (Page 1571)

AlarmStatisticsSettings.BeginTime

Description

The "BeginTime" property specifies the date and time for the start time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
AlarmStatisticsSettings.BeginTime
```

See also

AlarmControl.AlarmStatisticsSettings (Page 1577)

AlarmStatisticsSettings.MaximumRecords

Description

The "MaximumRecords" property specifies the maximum number of logged alarms.

Type

UInt16

Access

Read-write

Syntax

```
AlarmStatisticsSettings.MaximumRecords
```

See also

AlarmControl.AlarmStatisticsSettings (Page 1577)

AlarmStatisticsSettings.TimeRangeBase

Description

The "TimeRangeBase" property specifies the base of the time range.

Type

Int32, HmiTimeRangeBase

Specifies a time range:

- Undefined (0): Not defined
- Millisecond (1): Millisecond
- Second (2): Second
- Minute (3): Minute
- Hour (4): Hour
- Day (5): Day
- Month (6): Month
- Year (7): Year

Access

Read-write

Syntax

```
AlarmStatisticsSettings.TimeRangeBase
```

See also

AlarmControl.AlarmStatisticsSettings (Page 1577)

AlarmStatisticsSettings.TimeRangeFactor

Description

The "TimeRangeFactor" property specifies the factor of the time base for defining the time range.

Type

Int32

Access

Read-write

Syntax

```
AlarmStatisticsSettings.TimeRangeFactor
```

See also

AlarmControl.AlarmStatisticsSettings (Page 1577)

AlarmStatisticsSettings.TimeRangeStart

Description

The "TimeRangeStart" property specifies the start of the time range.

Type

Int32, HmiTimeRangeStart

Specifies the start of the time period:

- Now (0): Time period starts at the current time
- Fixed (1): Time period starts at a fixed time ("BeginTime" property)

Access

Read-write

Syntax

```
AlarmStatisticsSettings.TimeRangeStart
```

See also

AlarmControl.AlarmStatisticsSettings (Page 1577)

AlarmControl.AlarmStatisticsView

Description

The "AlarmStatisticsView" property specifies an alarm window with columns and alarm rows for statistical calculations on logged alarms.

Type

Object, HmiDataGridViewPart (Page 1630)

Access

Read-write

Syntax

```
AlarmControl.AlarmStatisticsView
```

See also

DataGridView (Page 1630)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 1583)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 1583)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 1583)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 1583)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 1586)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

HmiDataGridColumnCollection (Page 1586)

AlarmControl.AlarmStatisticsView (Page 1580)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 1586)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 1588)

See also

HmiDataGridColumnCollection (Page 1586)

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn

Description

The "AlarmStatisticColumn" object represents a value column.

Object type

HmiAlarmStatisticColumnPart

Properties

The "AlarmStatisticColumn" object has the following properties:

- **AlarmStatisticBlock**
  Sets the property of the alarm that is displayed in the column.
- **AllowSort**
  Specifies whether the sorting of the column is permitted.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies display options for text and graphics.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the direction of the sorting.
- **SortOrder**
  Specifies the order of the sorting.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width of the column in DIU (Device Independent Unit).

Methods

--

AlarmStatisticColumn.AlarmStatisticBlock

Description

The "AlarmStatisticBlock" property specifies the property of the alarm that is displayed in the column.

Type

Int32, HmiAlarmStatisticBlock

Specifies the alarm property:

- Undefined (0): Not defined
- AverageRaisedRaised (4097): Average time between the alarm and the resulting alarms
- AverageRaisedCleared (4098): Average time between the alarm and its clearance
- AverageRaisedAcknowledged (4099): Average time between the alarm and its acknowledgment
- AverageRaisedReset (4100): Average time between the alarm and its reset
- Frequency (4101): Number of alarms per unit of time
- SumRaisedRaised (4102): Sum of all alarms and resulting alarms
- SumRaisedCleared (4103): Sum of all alarms and their elimination
- SumRaisedAcknowledged (4104): Sum of all alarms and their acknowledgement
- SumRaisedReset (4105): Sum of all alarms and their resets
- ID (1): Alarm number
- Name (2): Alarm name
- Class (3): Alarm class
- Priority (4): Priority
- Group (5): Alarm group
- Origin (6): Origin
- Area (7): Area
- Comments (8): Alarm comment
- Information (9): Information text
- LoopInAlarm (10): Navigates to the screen in which the alarm was triggered.
- EventText (11): Alarm text
- AlarmText1...9 (12-20): User-defined alarm text
- AlarmState (21): Alarm state
- ModificationTime (22): Time of modification
- RaiseTime (23): Tripping time
- AcknowledgeTime (24): Time of acknowledgment

- ClearTime (25): Time of completion
- ResetTime (26): Reset time
- SuppressionState (27): Status of the alarm suppression
- EscalationLevel (28): Escalation level
- Context (29): Context
- Duration (30): Duration
- AcknowledgmentState (31): Acknowledgment state
- Value (32): Value
- ValueQuality (33): Quality code
- ValueLimit (34): Limit
- TagName (35): Name of the trigger tag
- HostName (36): PC name
- UserName (37): Logged-on user
- ProcessValue1...10 (38-47): Process value
- ClassSymbol (48): Alarm class icon
- StateText (49): Status text

Access

Read-write

Syntax

```
AlarmStatisticColumn.AlarmStatisticBlock
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.AllowSort

Description

The "AllowSort" property specifies whether the sorting of the column is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
AlarmStatisticColumn.AllowSort
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
AlarmStatisticColumn.BackColor
```

See also

AlarmStatisticColumn (Page 1588)

AlarmColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
AlarmStatisticColumn.Content
```

See also

AlarmStatisticColumn (Page 1588)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

AlarmColumn.Content (Page 1592)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

AlarmColumn.Content (Page 1592)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

AlarmColumn.Content (Page 1592)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

AlarmColumn.Content (Page 1592)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

AlarmColumn.Content (Page 1592)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

AlarmColumn.Content (Page 1592)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

AlarmColumn.Content (Page 1592)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

AlarmColumn.Content (Page 1592)

AlarmStatisticColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
AlarmStatisticColumn.Enabled
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
AlarmStatisticColumn.ForeColor
```

See also

AlarmStatisticColumn (Page 1588)

AlarmColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
AlarmStatisticColumn.Header
```

See also

AlarmStatisticColumn (Page 1588)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

AlarmColumn.Header (Page 1599)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 1599)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 1599)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

AlarmColumn.Header (Page 1599)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

AlarmColumn.Header (Page 1599)

AlarmStatisticColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
AlarmStatisticColumn.MaximumWidth
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
AlarmStatisticColumn.MinimumWidth
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
AlarmStatisticColumn.Name
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
AlarmStatisticColumn.OutputFormat
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.SortDirection

Description

The "SortDirection" property specifies the direction of the sorting.

Type

Int32, HmiSortDirection

Specifies the sorting order.

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
AlarmStatisticColumn.SortDirection
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.SortOrder

Description

The "SortOrder" property specifies the order of the sorting.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
AlarmStatisticColumn.SortOrder
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
AlarmStatisticColumn.Visible
```

See also

AlarmStatisticColumn (Page 1588)

AlarmStatisticColumn.Width

Description

The "Width" property specifies the width of the column in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
AlarmStatisticColumn.Width
```

See also

AlarmStatisticColumn (Page 1588)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 1610)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 1610)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 1610)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 1610)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 1610)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 1610)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 1617)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 1617)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 1622)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 1622)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 1622)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 1622)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 1622)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 1622)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

AlarmControl.AlarmStatisticsView (Page 1580)

AlarmControl.AlarmView

Description

The "AlarmView" property specifies the alarm window with columns and alarm lines for alarms without statistical evaluation.

Type

Object, HmiDataGridViewPart (Page 1630)

Access

Read-write

Syntax

```
AlarmControl.AlarmView
```

See also

AlarmControl (Page 1571)

DataGridView (Page 1630)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

AlarmControl.AlarmView (Page 1630)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 1633)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 1633)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 1633)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 1633)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 1636)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

HmiDataGridColumnCollection (Page 1636)

AlarmControl.AlarmView (Page 1630)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 1636)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 1638)

See also

HmiDataGridColumnCollection (Page 1636)

AlarmColumn (Page 1638)

AlarmColumn

Description

The "AlarmColumn" object represents a value column.

Object type

HmiAlarmColumnPart

Properties

The "AlarmColumn" object has the following properties:

- **AlarmBlock**
  Specifies which property of the alarm is displayed.
- **AllowSort**
  Specifies whether the sorting of the column is permitted.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies display options for text and graphics.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the direction of the sorting.
- **SortOrder**
  Specifies the order of the sorting.
- **UseAlarmColors**
  Specifies whether the configured alarm colors are used.

- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width of the column in DIU (Device Independent Unit).

Methods

--

AlarmColumn.AlarmBlock

Description

The "AlarmBlock" property specifies which property of the alarm is displayed.

Type

Int32, HmiAlarmBlock

Specifies the alarm property:

- Undefined (0): Not defined
- ID (1): Alarm number
- Name (2): Alarm name
- Class (3): Alarm class
- Priority (4): Priority
- Group (5): Alarm group
- Origin (6): Origin
- Area (7): Area
- Comments (8): Alarm comment
- Information (9): Information text
- LoopInAlarm (10): Navigates to the screen in which the alarm was triggered.
- EventText (11): Alarm text
- AlarmText1...9 (12-20): User-defined alarm text
- AlarmState (21): Alarm state
- ModificationTime (22): Time of modification
- RaiseTime (23): Tripping time
- AcknowledgeTime (24): Time of acknowledgment
- ClearTime (25): Time of completion
- ResetTime (26): Reset time
- SuppressionState (27): Status of the alarm suppression

- EscalationLevel (28): Escalation level
- Context (29): Context
- Duration (30): Duration
- AcknowledgmentState (31): Acknowledgment state
- Value (32): Value
- ValueQuality (33): Quality code
- ValueLimit (34): Limit
- HostName (36): PC name
- UserName (37): Logged-on user
- ProcessValue1...10 (38-47): Process value
- ClassSymbol (48): Alarm class icon
- StateText (49): Status text
- GroupID (50): Number of the alarm group

Access

Read-write

Syntax

```
AlarmColumn.AlarmBlock
```

See also

AlarmColumn (Page 1638)

AlarmColumn.AllowSort

Description

The "AllowSort" property specifies whether the sorting of the column is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
AlarmColumn.AllowSort
```

See also

AlarmColumn (Page 1638)

AlarmColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
AlarmColumn.BackColor
```

See also

AlarmColumn (Page 1638)

AlarmColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
AlarmColumn.Content
```

See also

AlarmColumn (Page 1638)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

AlarmColumn.Content (Page 1641)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

AlarmColumn.Content (Page 1641)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

AlarmColumn.Content (Page 1641)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

AlarmColumn.Content (Page 1641)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

AlarmColumn.Content (Page 1641)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

AlarmColumn.Content (Page 1641)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

AlarmColumn.Content (Page 1641)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

AlarmColumn.Content (Page 1641)

AlarmColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
AlarmColumn.Enabled
```

See also

AlarmColumn (Page 1638)

AlarmColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
AlarmColumn.ForeColor
```

See also

AlarmColumn (Page 1638)

AlarmColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
AlarmColumn.Header
```

See also

AlarmColumn (Page 1638)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

AlarmColumn.Header (Page 1648)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 1648)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 1648)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

AlarmColumn.Header (Page 1648)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

AlarmColumn.Header (Page 1648)

AlarmColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
AlarmColumn.MaximumWidth
```

See also

AlarmColumn (Page 1638)

AlarmColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
AlarmColumn.MinimumWidth
```

See also

AlarmColumn (Page 1638)

AlarmColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
AlarmColumn.Name
```

See also

AlarmColumn (Page 1638)

AlarmColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
AlarmColumn.OutputFormat
```

See also

AlarmColumn (Page 1638)

AlarmColumn.SortDirection

Description

The "SortDirection" property specifies the direction of the sorting.

Type

Int32, HmiSortDirection

Specifies the sorting order.

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
AlarmColumn.SortDirection
```

See also

AlarmColumn (Page 1638)

AlarmColumn.SortOrder

Description

The "SortOrder" property specifies the order of the sorting.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
AlarmColumn.SortOrder
```

See also

AlarmColumn (Page 1638)

AlarmColumn.UseAlarmColors

Description

The "UseAlarmColors" property specifies whether the configured alarm colors are used.

Type

Bool

Access

Read-write

Syntax

```
AlarmColumn.UseAlarmColors
```

See also

AlarmColumn (Page 1638)

AlarmColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
AlarmColumn.Visible
```

See also

AlarmColumn (Page 1638)

AlarmColumn.Width

Description

The "Width" property specifies the width of the column in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
AlarmColumn.Width
```

See also

AlarmColumn (Page 1638)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

AlarmControl.AlarmView (Page 1630)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 1660)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 1660)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 1660)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 1660)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 1660)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 1660)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`
Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 1666)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 1666)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 1671)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 1671)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 1671)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 1671)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 1671)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 1671)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

AlarmControl.AlarmView (Page 1630)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

AlarmControl.AlarmView (Page 1630)

AlarmControl.AlwaysShowRecent

Description

The "AlwaysShowRecent" specifies whether the most recent alarm is displayed at the beginning or end of the list, depending on the sorting.

Type

Bool

Access

Read-write

Syntax

```
AlarmControl.AlwaysShowRecent
```

See also

AlarmControl (Page 1571)

AlarmControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
AlarmControl.BackColor
```

See also

AlarmControl (Page 1571)

AlarmControl.Caption

Description

The "Caption" property specifies the text to be displayed in the header.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
AlarmControl.Caption
```

See also

AlarmControl (Page 1571)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

AlarmControl.Caption (Page 1680)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 1681)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 1681)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 1681)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 1681)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 1681)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 1681)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

AlarmControl.Caption (Page 1680)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

AlarmControl.Caption (Page 1680)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

AlarmControl.Caption (Page 1680)

AlarmControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the header.

Type

UInt32

Access

Read-write

Syntax

```
AlarmControl.CaptionColor
```

See also

AlarmControl (Page 1571)

AlarmControl.CurrentQuality

Description

The "CurrentQuality" property returns the current worst quality code of all tags which influence the alarm control.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable. Quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
AlarmControl.CurrentQuality
```

See also

AlarmControl (Page 1571)

AlarmControl.DefaultSortDirection

Description

The "DefaultSortDirection" property specifies the sorting order of the time column if no other sorting is active.

Type

Int32, HmiSortDirection

Specifies the sorting order.

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
AlarmControl.DefaultSortDirection
```

See also

AlarmControl (Page 1571)

AlarmControl.Enabled

Description

The "Enabled" property specifies whether the alarm control can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
AlarmControl.Enabled
```

See also

AlarmControl (Page 1571)

AlarmControl.Filter

Description

The "Filter" property specifies a string for filtering alarms.

The syntax of the filter string corresponds to the WHERE clause of an SQL command.

Type

String

Access

Read-write

Syntax

```
AlarmControl.Filter
```

Supported alarm properties

The following properties of an alarm can be used in the filter string:

- AcknowledgementTime
- Alarm
- AlarmClassName
- AlarmClassSymbol
- AlarmParameterValues
- AlarmText1 ... 9
- Area
- BackColor
- ChangeReason
- ClearTime
- Connection
- EventText
- Flashing
- InfoText
- InstanceID
- LoopInAlarm
- ModificationTime
- Name
- Origin
- Priority
- RaiseTime
- ResetTime
- SourceID
- SourceType
- State
- StateMachine
- StateText
- SuppressionState
- SystemSeverity
- TextColor
- Value
- ValueLimit

Operators

The following operators can be used in the filter string:

Operator	Description	Example
=	equal to	AlarmClassName = 'demo'
IS NOT string	is not equal to the string *string*	AlarmText4 IS NOT 'Text5'
<>	not equal	Value <> 0.0
>	greater than	ModificationTime > '11.08.2016'
<	less than	Value < 75.0
>=	greater than or equal to	Value >= 25.0
<=	less than or equal to	Value <= 75.0
OR, ||	logical OR	State = 1 OR State = 3
AND, &&	logical AND	EventText = 'Text1' AND Origin = 'Motor'
BETWEEN	within a range	Value BETWEEN 25.0 AND 75.0
NOT BETWEEN	outside a range	Value NOT BETWEEN 25.0 AND 75.0
LIKE string	corresponds to the string *string*	Name LIKE 'Motor*'
NOT LIKE string	does not correspond to the string *string*	Name NOT LIKE 'Valve*'
IN (v1, v2, …)	corresponds to one or more values	State IN (1, 4, 7)
NOT IN (v1, v2, …)	does not correspond to one or more values	State NOT IN (0, 2, 3, 5, 6)
IS NULL	compares to zero (missing data)	Context IS NULL
IS NOT NULL	compares to zero (unknown data)	Context IS NOT NULL

Wildcards

The following wildcards can be used for characters of filter strings:

Wildcard	Description	Example
*	replaces 0, 1 or more characters	Name LIKE 'Motor*'
?	replaces exactly 1 character	Name = 'Recipe?'

See also

AlarmControl (Page 1571)

AlarmControl.Height

Description

The "Height" property specifies the height of the alarm control in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
AlarmControl.Height
```

See also

AlarmControl (Page 1571)

AlarmControl.Icon

Description

The "Icon" property specifies the icon of the alarm control.

Type

String

Access

Read-write

Syntax

```
AlarmControl.Icon
```

See also

AlarmControl (Page 1571)

AlarmControl.Layer

Description

The "Layer" property returns the layer of the screen in which the alarm control is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
AlarmControl.Layer
```

See also

AlarmControl (Page 1571)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

AlarmControl.Layer (Page 1691)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

AlarmControl.Layer (Page 1691)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

AlarmControl.Layer (Page 1691)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

AlarmControl.Layer (Page 1691)

AlarmControl.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
AlarmControl.Left
```

See also

AlarmControl (Page 1571)

AlarmControl.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the alarm control.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
AlarmControl.Margin
```

See also

AlarmControl (Page 1571)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

AlarmControl.Margin (Page 1694)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

AlarmControl.Margin (Page 1694)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

AlarmControl.Margin (Page 1694)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

AlarmControl.Margin (Page 1694)

AlarmControl.Name

Description

The "Name" property returns the name of the alarm control.

Type

String

Access

Read-only

Syntax

```
AlarmControl.Name
```

See also

AlarmControl (Page 1571)

AlarmControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
AlarmControl.Parent
```

See also

AlarmControl (Page 1571)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

AlarmControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the object has been created.

Type

String

Access

Read-only

Syntax

```
AlarmControl.RenderingTemplate
```

See also

AlarmControl (Page 1571)

AlarmControl.ResetFlashingRate

Description

The "ResetFlashingRate" property specifies the flashing frequency for alarms that need to be reset.

Type

Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Access

Read-write

Syntax

```
AlarmControl.ResetFlashingRate
```

See also

AlarmControl (Page 1571)

AlarmControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the alarm window is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
AlarmControl.ShowFocusVisual
```

See also

AlarmControl (Page 1571)

AlarmControl.StatusBar

Description

The "StatusBar" property specifies the Information bar of the alarm control.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
AlarmControl.StatusBar
```

See also

AlarmControl (Page 1571)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

AlarmControl.StatusBar (Page 1700)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 1701)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

AlarmControl.StatusBar (Page 1700)

HmiControlBarElementCollection (Page 1701)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 1701)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 1720)

See also

HmiControlBarElementCollection (Page 1701)

Control Bar Elements (Page 1720)

Control Bar Elements

Description

Control Bar Elements (Page 1720)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

AlarmControl.StatusBar (Page 1700)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

AlarmControl.StatusBar (Page 1700)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 1704)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 1704)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 1704)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 1704)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 1704)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 1704)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

AlarmControl.StatusBar (Page 1700)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 1708)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 1708)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 1708)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 1708)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

AlarmControl.StatusBar (Page 1700)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 1710)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 1710)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 1710)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 1710)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

AlarmControl.StatusBar (Page 1700)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

AlarmControl.StatusBar (Page 1700)

AlarmControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the alarm control.

Type

String

Access

Read-only

Syntax

```
AlarmControl.StyleItemClass
```

See also

AlarmControl (Page 1571)

AlarmControl.SuppressFlashing

Description

The "SuppressFlashing" property suppresses all flashing of the alarm control.

Type

Bool

Access

Read-write

Syntax

```
AlarmControl.SuppressFlashing
```

See also

AlarmControl (Page 1571)

AlarmControl.Systems

Description

The "Systems" property specifies the name of the runtime system for compiling active alarms.

If no reference is specified, all known systems are used.

Type

String[] | Int32[]

Access

Read-write

Syntax

```
AlarmControl.Systems
```

See also

AlarmControl (Page 1571)

AlarmControl.TabIndex

Description

The "TabIndex" property returns the position of the alarm control in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
AlarmControl.TabIndex
```

See also

AlarmControl (Page 1571)

AlarmControl.TimeZone

Description

The "TimeZone" property specifies the time zone.

Type

Int32, HmiTimeZone

Access

Read-write

Syntax

```
AlarmControl.TimeZone
```

See also

AlarmControl (Page 1571)

AlarmControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the alarm control.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
AlarmControl.ToolBar
```

See also

AlarmControl (Page 1571)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

AlarmControl.ToolBar (Page 1716)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 1718)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

AlarmControl.ToolBar (Page 1716)

HmiControlBarElementCollection (Page 1718)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 1718)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 1720)

See also

HmiControlBarElementCollection (Page 1718)

Control Bar Elements (Page 1720)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.

- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 1720)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 1720)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 1720)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 1720)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 1720)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 1725)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 1725)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 1725)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 1725)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 1725)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above

- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 1725)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 1725)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 1725)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 1720)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 1720)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog

- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled

- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 1720)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 1735)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 1735)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 1735)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 1735)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 1720)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 1720)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 1720)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 1720)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 1741)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 1741)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 1741)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 1741)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 1720)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 1720)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 1720)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 1746)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 1746)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 1748)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 1748)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 1748)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 1748)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 1748)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 1748)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 1748)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 1748)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 1746)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 1746)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 1758)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 1758)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 1758)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 1758)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 1746)

ControlBarButton.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 1746)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 1763)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 1763)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 1763)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 1763)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 1746)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 1746)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 1768)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 1775)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 1775)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 1775)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 1775)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 1768)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 1780)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 1780)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 1780)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 1780)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 1768)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 1768)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 1785)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 1791)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 1791)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 1791)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 1791)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 1785)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 1796)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 1796)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 1796)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 1796)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 1785)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 1785)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 1801)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 1810)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 1810)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 1810)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 1810)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 1801)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 1815)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 1815)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 1815)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 1815)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 1801)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 1801)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 1821)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 1828)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 1828)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 1821)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 1839)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 1839)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 1839)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 1839)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 1821)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 1844)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 1844)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 1844)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 1844)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 1821)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 1821)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

AlarmControl.ToolBar (Page 1716)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

AlarmControl.ToolBar (Page 1716)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 1850)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 1850)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 1850)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 1850)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 1850)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 1850)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

AlarmControl.ToolBar (Page 1716)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 1853)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 1853)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 1853)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 1853)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

AlarmControl.ToolBar (Page 1716)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 1856)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 1856)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 1856)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 1856)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

AlarmControl.ToolBar (Page 1716)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

AlarmControl.ToolBar (Page 1716)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

AlarmControl.ToolBar (Page 1716)

AlarmControl.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
AlarmControl.Top
```

See also

AlarmControl (Page 1571)

AlarmControl.UseAlarmColors

Description

The "UseAlarmColors" property specifies whether the configured alarm colors are used.

Type

Bool

Access

Read-write

Syntax

```
AlarmControl.UseAlarmColors
```

See also

AlarmControl (Page 1571)

AlarmControl.Visible

Description

The "Visible" property specifies whether the alarm control is visible.

Type

Bool

Access

Read-write

Syntax

```
AlarmControl.Visible
```

See also

AlarmControl (Page 1571)

AlarmControl.Width

Description

The "Width" property specifies the width of the alarm control in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
AlarmControl.Width
```

See also

AlarmControl (Page 1571)

AlarmControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the alarm control.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
AlarmControl.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

AlarmControl (Page 1571)

AlarmControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the alarm control.

Syntax

```
AlarmControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

AlarmControl (Page 1571)

AlarmControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the alarm control.

Syntax

```
AlarmControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

AlarmControl (Page 1571)

AlarmControl.GetSelectedAlarmData()

Description

The "GetSelectedAlarmData" method returns the visible data of the selected alarm of the alarm control.

Syntax

```
AlarmControl.GetSelectedAlarmData()
```

Parameters

--

Return value

--

See also

AlarmControl (Page 1571)

AlarmControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
AlarmControl.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

AlarmControl (Page 1571)

AlarmControl_OnActivated()

Description

The "OnActivated" event occurs when an alarm control receives focus:

- An alarm control is selected via the configured tab sequence.
- An alarm control that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and receives focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
AlarmControl_OnActivated(item)
```

Context

`item`

Type: Object

Alarm control at which the event occurs.

See also

AlarmControl (Page 1571)

AlarmControl_OnDeactivated() (Page 1867)

AlarmControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when an alarm control loses focus when the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

Syntax

```
AlarmControl_OnDeactivated(item)
```

Context

item

Type: Object

Alarm control at which the event occurs.

See also

AlarmControl (Page 1571)

AlarmControl_OnActivated() (Page 1866)

AlarmControl_OnInitialized()

Description

The "OnInitialized" event occurs when an alarm control has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
AlarmControl_OnInitialized(item)
```

Context

item

Type: Object

Alarm control at which the event occurs.

See also

AlarmControl (Page 1571)

AlarmControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the alarm control.

Syntax

```
AlarmControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Alarm control at which the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

AlarmControl (Page 1571)

Control Bar Elements (Page 1720)

AlarmControl_OnSelectionChanged()

Description

The "OnSelectionChanged" event occurs when the selection in the alarm control changes.

Syntax

```
AlarmControl_OnSelectionChanged(item, SelectedRowData)
```

Context

item

Type: Object

Alarm control at which the event occurs.

SelectedRowData

Type: Variant

Visible properties of the selected alarm row of the alarm control.

See also

AlarmControl (Page 1571)

Bar

Description

The "Bar" object represents a bar for the display of process values in runtime.

Object type

HmiBar

Properties

The "Bar" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.

- **BackColor**
  Specifies the background color.
- **BarMode**
  Specifies the color of the bar.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **ComputedMaxPeakValue**
  Returns the highest process value that occurred.
- **ComputedMinPeakValue**
  Returns the lowest process value that occurred.
- **ComputedValueTendency**
  Returns the change of the process value.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the bar.
- **Enabled**
  Specifies whether the bar can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **Height**
  Specifies the height.
- **Label**
  Specifies the labeling below the bar.
- **Layer**
  Returns the layer of the screen in which the bar is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the bar.
- **NormalRangeColor**
  Specifies the color of the normal range.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the bar is operable.
- **OriginValue**
  Specifies the output value of the normal range that is visualized.
- **OutputFormat**
  Specifies the format for displaying the process values.

- **Parent**
  Returns the higher-level screen object.
- **PeakIndicators**
  Specifies whether the highest and lowest process value up to this time are displayed.
- **ProcessValue**
  Specifies the process value.
- **ProcessValueIndicatorBackColor**
  Specifies the background color for the process value.
- **ProcessValueIndicatorForeColor**
  Specifies the foreground color for the process value.
- **ProcessValueIndicatorMode**
  Specifies the type of display of the current process value.
- **RelativeToOrigin**
  Specifies whether the output value is an absolute or a percentage value between the minimum and maximum value.
- **RenderingTemplate**
  Returns the name of the template from which the bar was created.
- **RequireExplicitUnlock**
  Returns whether the bar is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the bar rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ScaleBackColor**
  Specifies the background color of the scale.
- **ScaleForeColor**
  Specifies the foreground color of the scale.
- **ShowFocusVisual**
  Specifies whether the bar is highlighted when in focus.
- **ShowTrendIndicator**
  Specifies whether the tendency (rising or falling) of the process value to be monitored is indicated by means of a small arrow.
- **StraightScale**
  Specifies the scale of the bar.
- **StyleItemClass**
  Returns the style which is applied to the bar.
- **TabIndex**
  Returns the position of the bar in the tab sequence.

- **ThresholdIndicators**
  Specifies how parameterized limit values are visualized.
- **Thresholds**
  Returns the list of all limit values of the bar.
- **Title**
  Specifies the caption which appears as the title.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **TrendIndicatorColor**
  Specifies the color of the trend indicator.
- **Visible**
  Specifies whether the bar is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "Bar" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the bar.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Bar" object has the following events:

- **OnActivated()**
  Occurs when a bar receives focus.
- **OnContextTapped()**
  Occurs when a bar is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a bar loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the bar is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the bar is in focus.
- **OnTapped()**
  Occurs when a bar is left-clicked or short-touched.

Bar.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Bar.AlternateBackColor
```

See also

Bar (Page 1869)

Bar.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Bar.AlternateBorderColor
```

See also

Bar (Page 1869)

Bar.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Bar.Authorization
```

See also

Bar (Page 1869)

Bar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Bar.BackColor
```

See also

Bar (Page 1869)

Bar.BarMode

Description

The "BarMode" property specifies the color of the bar.

Type

Int32, HmiBarMode

Specifies the bar mode:

- Segmented (0): Bar changes color according to the bar segments.
- Unicolor (1): Entire bar has same color.
- SegmentedStatic (2): Bar segments in the background, process value indicator in front of the bar segments.
- UnicolorStatic (3): Background color changes according to the process value and the limit colors, process value indicator runs in front of the bar segments.

Access

Read-write

Syntax

```
Bar.BarMode
```

See also

Bar (Page 1869)

Bar.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Bar.BorderColor
```

See also

Bar (Page 1869)

Bar.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Bar.BorderWidth
```

See also

Bar (Page 1869)

Bar.ComputedMaxPeakValue

Description

The "ComputedMaxPeakValue" property returns the highest process value that occurred.

Type

Variant

Access

Read-only

Syntax

```
Bar.ComputedMaxPeakValue
```

See also

Bar (Page 1869)

Bar.ComputedMinPeakValue (Page 1877)

Bar.ComputedMinPeakValue

Description

The "ComputedMinPeakValue" property returns the lowest process value which occurred.

Type

Variant

Access

Read-only

Syntax

```
Bar.ComputedMinPeakValue
```

See also

Bar (Page 1869)

Bar.ComputedMaxPeakValue (Page 1876)

Bar.ComputedValueTendency

Description

The "ComputedValueTendency" property returns the change in the process value.

Type

Int32, HmiValueTendency

Returns the modification:

- Steady (0): No change
- Upwards (1): Change upwards
- Downwards (2): Change downwards

Access

Read-only

Syntax

```
Bar.ComputedValueTendency
```

See also

Bar (Page 1869)

Bar.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the bar.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Bar.CurrentQuality
```

See also

Bar (Page 1869)

Bar.Enabled

Description

The "Enabled" property specifies whether the bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Bar.Enabled
```

See also

Bar (Page 1869)

Bar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Bar.Font
```

See also

Bar (Page 1869)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Bar.Font (Page 1879)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Bar.Font (Page 1879)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Bar.Font (Page 1879)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Bar.Font (Page 1879)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Bar.Font (Page 1879)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Bar.Font (Page 1879)

Bar.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Bar.Height
```

See also

Bar (Page 1869)

Bar.Label

Description

The "Label" property specifies the label below the bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Bar.Label
```

See also

Bar (Page 1869)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Bar.Label (Page 1883)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 1884)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 1884)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 1884)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 1884)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 1884)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 1884)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Bar.Label (Page 1883)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Bar.Label (Page 1883)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Bar.Label (Page 1883)

Bar.Layer

Description

The "Layer" property returns the layer of the screen in which the bar is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Bar.Layer
```

See also

Bar (Page 1869)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Bar.Layer (Page 1889)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Bar.Layer (Page 1889)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Bar.Layer (Page 1889)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Bar.Layer (Page 1889)

Bar.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Bar.Left
```

See also

Bar (Page 1869)

Bar.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Bar.Margin
```

See also

Bar (Page 1869)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Bar.Margin (Page 1892)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Bar.Margin (Page 1892)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Bar.Margin (Page 1892)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Bar.Margin (Page 1892)

Bar.Name

Description

The "Name" property returns the name of the bar.

Type

String

Access

Read-only

Syntax

```
Bar.Name
```

See also

Bar (Page 1869)

Bar.NormalRangeColor

Description

The "NormalRangeColor" property specifies the color of the normal range.

Type

UInt32

Access

Read-write

Syntax

```
Bar.NormalRangeColor
```

See also

Bar (Page 1869)

Bar.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Bar.Opacity
```

See also

Bar (Page 1869)

Bar.Operability

Description

The "Operability" property returns whether the bar is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Bar.Operability
```

See also

Bar (Page 1869)

Bar.OriginValue

Description

The "OriginValue" property specifies the output value of the normal range to be visualized.

Type

Float

Access

Read-write

Syntax

```
Bar.OriginValue
```

See also

Bar (Page 1869)

Bar.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
Bar.OutputFormat
```

See also

Bar (Page 1869)

Bar.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Bar.Parent
```

See also

Bar (Page 1869)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Bar.PeakIndicators

Description

The "PeakIndicators" property specifies whether the highest and lowest process value up to this time are displayed.

Type

Int32, HmiPeakIndicator

Specifies the display of the peak indicator:

- None (0): No display
- Low (1): Only the lowest process value
- High (2): Only the highest process value

Access

Read-write

Syntax

```
Bar.PeakIndicators
```

See also

Bar (Page 1869)

Bar.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
Bar.ProcessValue
```

See also

Bar (Page 1869)

Bar.ProcessValueIndicatorBackColor

Description

The "ProcessValueIndicatorBackColor" property specifies the background color for the process value indicator.

Type

UInt32

Access

Read-write

Syntax

```
Bar.ProcessValueIndicatorBackColor
```

See also

Bar (Page 1869)

Bar.ProcessValueIndicatorForeColor (Page 1900)

Bar.ProcessValueIndicatorForeColor

Description

The "ProcessValueIndicatorForeColor" property specifies the foreground color for the process value indicator.

Type

UInt32

Access

Read-write

Syntax

```
Bar.ProcessValueIndicatorForeColor
```

See also

Bar (Page 1869)

Bar.ProcessValueIndicatorBackColor (Page 1899)

Bar.ProcessValueIndicatorMode

Description

The "ProcessValueIndicatorMode" property specifies the type of display of the current process value.

Type

Int32, HmiProcessIndicatorMode

Specifies the type of display:

- Bar (0): Bar only
- Indicator (1): Hair line or needle, no numerical display of the process value.
- DetailedIndicator (2): Detailed display with numerical value
- BarWithDetailedIndicator (3): Bar with numerical value

Access

Read-write

Syntax

```
Bar.ProcessValueIndicatorMode
```

See also

Bar (Page 1869)

Bar.RelativeToOrigin

Description

The "RelativeToOrigin" property specifies whether the output value is an absolute or a percentage value between minimum and maximum value.

Type

Bool

Access

Read-write

Syntax

```
Bar.RelativeToOrigin
```

See also

Bar (Page 1869)

Bar.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the bar was created.

Type

String

Access

Read-only

Syntax

```
Bar.RenderingTemplate
```

See also

Bar (Page 1869)

Bar.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the bar can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Bar.RequireExplicitUnlock
```

See also

Bar (Page 1869)

Bar.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Bar.RotationAngle
```

See also

Bar (Page 1869)
Bar.RotationCenterPlacement (Page 1903)
Bar.RotationCenterX (Page 1904)
Bar.RotationCenterY (Page 1905)

Bar.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the bar rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Bar.RotationCenterPlacement
```

See also

Bar (Page 1869)

Bar.RotationAngle (Page 1903)

Bar.RotationCenterX (Page 1904)

Bar.RotationCenterY (Page 1905)

Bar.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Bar.RotationCenterX
```

See also

Bar (Page 1869)

Bar.RotationAngle (Page 1903)

Bar.RotationCenterPlacement (Page 1903)

Bar.RotationCenterY (Page 1905)

Bar.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Bar.RotationCenterY
```

See also

Bar (Page 1869)

Bar.RotationAngle (Page 1903)

Bar.RotationCenterPlacement (Page 1903)

Bar.RotationCenterX (Page 1904)

Bar.ScaleBackColor

Description

The "ScaleBackColor" property specifies the background color of the scale.

Type

UInt32

Access

Read-write

Syntax

```
Bar.ScaleBackColor
```

See also

Bar (Page 1869)

Bar.ScaleForeColor (Page 1906)

Bar.ScaleForeColor

Description

The "ScaleForeColor" property specifies the foreground color of the scale.

Type

UInt32

Access

Read-write

Syntax

```
Bar.ScaleForeColor
```

See also

Bar (Page 1869)

Bar.ScaleBackColor (Page 1905)

Bar.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the bar is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Bar.ShowFocusVisual
```

See also

Bar (Page 1869)

Bar.ShowTrendIndicator

Description

The "ShowTrendIndicator" property specifies whether the tendency (rising or falling) of the process value to be monitored is indicated by means of a small arrow.

Type

Bool

Access

Read-write

Syntax

```
Bar.ShowTrendIndicator
```

See also

Bar (Page 1869)

Bar.StraightScale

Description

The "StraightScale" property specifies the scale of the bar.

Type

Object, HmiStraightScalePart

Access

Read-write

Syntax

```
Bar.StraightScale
```

See also

Bar (Page 1869)

StraightScale.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
StraightScale.AutoScaling
```

See also

Bar.StraightScale (Page 1907)

StraightScale.BeginValue

Description

The "BeginValue" property specifies the start of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
StraightScale.BeginValue
```

See also

Bar.StraightScale (Page 1907)

StraightScale.DivisionCount

Description

The "DivisionCount" property specifies the number of main units with subdivisions. To this purpose the automatic scaling must be switched off.

Type

Int32

Access

Read-write

Syntax

```
StraightScale.DivisionCount
```

See also

Bar.StraightScale (Page 1907)

StraightScale.EndValue

Description

The "EndValue" property specifies the end of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
StraightScale.EndValue
```

See also

Bar.StraightScale (Page 1907)

StraightScale.LabelColor

Description

The "LabelColor" property specifies the color of the labeling.

Type

UInt32

Access

Read-write

Syntax

```
StraightScale.LabelColor
```

See also

Bar.StraightScale (Page 1907)

StraightScale.LabelFont

Description

The "LabelFont" property specifies the font of the labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StraightScale.LabelFont
```

See also

Bar.StraightScale (Page 1907)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StraightScale.LabelFont (Page 1910)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StraightScale.LabelFont (Page 1910)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StraightScale.LabelFont (Page 1910)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StraightScale.LabelFont (Page 1910)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StraightScale.LabelFont (Page 1910)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StraightScale.LabelFont (Page 1910)

StraightScale.LargeTickLabelingStep

Description

The "LargeTickLabelingStep" property specifies the interval at which scale sections are labeled.

Type

UInt8

Access

Read-write

Syntax

```
StraightScale.LargeTickLabelingStep
```

See also

Bar.StraightScale (Page 1907)

StraightScale.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
StraightScale.MeasurementUnit
```

See also

Bar.StraightScale (Page 1907)

StraightScale.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
StraightScale.MeasurementUnitType
```

See also

Bar.StraightScale (Page 1907)

StraightScale.Orientation

Description

The "Orientation" property specifies the orientation of the scale.

Type

Int32, HmiOrientation

Specifies the alignment:

- Horizontal (0): Horizontal
- Vertical (1): Vertical

Access

Read-write

Syntax

```
StraightScale.Orientation
```

See also

Bar.StraightScale (Page 1907)

StraightScale.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
StraightScale.OutputFormat
```

See also

Bar.StraightScale (Page 1907)

StraightScale.ScaleMode

Description

The "ScaleMode" property specifies the type of scaling.

Type

Int32, HmiScaleMode

Specifies the scaling:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
StraightScale.ScaleMode
```

See also

Bar.StraightScale (Page 1907)

StraightScale.ScalingType

Description

The "ScalingType" property specifies the scaling.

Type

Int32, HmiScalingType

Specifies the scaling:

- Linear (0): Linear
- Logarithmic (1): Logarithmic
- NegativeLogarithmic (2): Negative logarithmic

- Tangent (4): Tangential
- Quadratic (5): Square
- Cubic (6): Cubic

Access

Read-write

Syntax

```
StraightScale.ScalingType
```

See also

Bar.StraightScale (Page 1907)

StraightScale.SubDivisionCount

Description

The "SubDivisionCount" property specifies the number of subdivisions of the main units.

Type

Int32

Access

Read-write

Syntax

```
StraightScale.SubDivisionCount
```

See also

Bar.StraightScale (Page 1907)

StraightScale.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
StraightScale.TickColor
```

See also

Bar.StraightScale (Page 1907)

Bar.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the bar.

Type

String

Access

Read-only

Syntax

```
Bar.StyleItemClass
```

See also

Bar (Page 1869)

Bar.TabIndex

Description

The "TabIndex" property returns the position of the bar in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Bar.TabIndex
```

See also

Bar (Page 1869)

Bar.ThresholdIndicators

Description

The "ThresholdIndicators" property specifies how parameterized limit values are visualized.

Type

Int32, HmiThresholdIndicator

Specifies the visualization:

- None (0): None
- Lines (1): Lines
- Markers (2): Markers

Access

Read-write

Syntax

```
Bar.ThresholdIndicators
```

See also

Bar (Page 1869)

Bar.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the bar.

Type

Object, HmiThresholdCollection (Page 1921)

Access

Read-only

Syntax

```
Bar.Thresholds
```

See also

Bar (Page 1869)

HmiThresholdCollection (Page 1921)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

Bar.Thresholds (Page 1920)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 1921)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 1923)

See also

HmiThresholdCollection (Page 1921)

Threshold (Page 1923)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.
- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

See also

HmiThresholdCollection (Page 1921)

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 1923)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 1923)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 1923)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold
- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 1923)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 1923)

Bar.Title

Description

The "Title" property specifies the caption that appears as the title.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Bar.Title
```

See also

Bar (Page 1869)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Bar.Title (Page 1926)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 1927)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 1927)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 1927)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 1927)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 1927)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 1927)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Bar.Title (Page 1926)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Bar.Title (Page 1926)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Bar.Title (Page 1926)

Bar.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Bar.ToolTipText
```

See also

Bar (Page 1869)

Bar.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Bar.Top
```

See also

Bar (Page 1869)

Bar.TrendIndicatorColor

Description

The "TrendIndicatorColor" property specifies the color of the trend indicator. The trend indicator uses a small arrow to represent the tendency (rising or falling) of the process value to be monitored. To activate the trend indicator, the "ShowTrendIndicator" property must be activated.

Type

UInt32

Access

Read-write

Syntax

```
Bar.TrendIndicatorColor
```

See also

Bar (Page 1869)

Bar.Visible

Description

The "Visible" property specifies whether the bar is visible.

Type

Bool

Access

Read-write

Syntax

```
Bar.Visible
```

See also

Bar (Page 1869)

Bar.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
Bar.VisualizeQuality
```

See also

Bar (Page 1869)

Bar.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Bar.Width
```

See also

Bar (Page 1869)

Bar.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the bar.

Syntax

```
Bar.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Bar (Page 1869)

Bar.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Bar.PropertyFlashing(propertyName, enable[, value][, alternateValue]
[, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Bar (Page 1869)

Bar_OnActivated()

Description

The "OnActivated" event occurs when a bar receives focus:

- A bar is selected via the configured tab sequence.
- A bar that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Bar_OnActivated(item)
```

Context

item

Type: Object

Bar where the event occurs.

See also

Bar (Page 1869)

Bar_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A bar is right-clicked.
- A bar is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Bar_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Bar where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Bar (Page 1869)

Bar_OnDeactivated()

Description

The "OnDeactivated" event occurs when a bar loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Bar_OnDeactivated(item)
```

Context

item

Type: Object

Bar where the event occurs.

See also

Bar (Page 1869)

Bar_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the bar is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Bar_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Bar where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Bar (Page 1869)

Bar_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the bar is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Bar_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Bar where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Bar (Page 1869)

Bar_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A bar is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a bar has the focus.
- A bar is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Bar_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Bar where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Bar (Page 1869)

BindingSourceElement

Description

The "BindingSourceElement" object represents a data source in runtime.

Object type

HmiBindingSourceElement

Properties

The "BindingSourceElement" object has the following properties:

- **AutoRequery**
  Specifies whether the data source is queried automatically.
- **ConsideredColumns**
  Returns information about the selected columns.
- **CursorPosition**
  Specifies the position of the cursor.
- **DataConnection**
  Returns the data connection.
- **Parent**
  Returns the higher-level screen object.
- **ReadCommand**
  Returns information about the command.
- **RowCount**
  Returns the row count.
- **SourceState**
  Returns the state of the data source.

Methods

The "BindingSourceElement" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the combo box.
- **MoveFirst()**
  Moves the data source to the first position.
- **MoveLast()**
  Moves the data source to the last position.
- **MoveNext()**
  Moves the data source forwards.
- **MovePrevious()**
  Moves the data source backwards.
- **PropertyFlashing()**
  Configures flashing of a property.

BindingSourceElement.AutoRequery

Description

The "AutoRequery" property specifies whether the data source is queried automatically.

Type

Bool

Access

Read-write

Syntax

```
BindingSourceElement.AutoRequery
```

See also

BindingSourceElement (Page 1943)

BindingSourceElement.ConsideredColumns

Description

The "ConsideredColumns" returns information about the selected columns.

Type

Object, HmiConsideredColumnCollection (Page 1945)

Access

Read-only

Syntax

```
BindingSourceElement.ConsideredColumns
```

See also

BindingSourceElement (Page 1943)

HmiConsideredColumnCollection (Page 1945)

HmiConsideredColumnCollection

Description

The "HmiConsideredColumnCollection" object is a list of all columns ("ConsideredColumn" objects).

Use

The "HmiConsideredColumnCollection" object is a list which can be counted and enumerated. You can access the "HmiConsideredColumnCollection" list using the index or the tag names.

Object type

HmiConsideredColumnCollection

Properties

The "HmiConsideredColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiConsideredColumnCollection" list.

Methods

The "HmiConsideredColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiConsideredColumnCollection" list.

See also

BindingSourceElement.ConsideredColumns (Page 1945)

HmiConsideredColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiConsideredColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiConsideredColumnCollection.Count
```

See also

HmiConsideredColumnCollection (Page 1945)

HmiConsideredColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiConsideredColumnCollection" list.

Syntax

```
HmiConsideredColumnCollection.[.Item](HmiConsideredColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiConsideredColumnCollection" object.

Parameters

`HmiConsideredColumnName`

Type: String

Name of the column

Return value

Object, HmiConsideredColumnPart (Page 1947)

See also

HmiConsideredColumnCollection (Page 1945)

ConsideredColumn (Page 1947)

ConsideredColumn

Description

The "ConsideredColumn" object represents a column.

Object type

HmiConsideredColumnPart

Properties

The "ConsideredColumn" object has the following properties:

- **IsPrimary**
  Returns whether the column is the primary column.
- **Key**
  Returns the key of the column.

Methods

--

See also

HmiConsideredColumnCollection (Page 1945)

ConsideredColumn.IsPrimary

Description

The "IsPrimary" property returns whether the column is the primary column.

Type

Bool

Access

Read-only

Syntax

```
ConsideredColumn.IsPrimary
```

See also

ConsideredColumn (Page 1947)

ConsideredColumn.Key

Description

The "Key" property returns the key of the column.

Type

String

Access

Read-only

Syntax

```
ConsideredColumn.Key
```

See also

ConsideredColumn (Page 1947)

BindingSourceElement.CursorPosition

Description

The "CursorPosition" property specifies the position of the cursor.

Type

UInt32

Access

Read-write

Syntax

```
BindingSourceElement.CursorPosition
```

See also

BindingSourceElement (Page 1943)

BindingSourceElement.DataConnection

Description

The "DataConnection" property returns the data connection.

Type

String, DataConnection

Access

Read-only

Syntax

```
BindingSourceElement.DataConnection
```

See also

BindingSourceElement (Page 1943)

BindingSourceElement.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
BindingSourceElement.Parent
```

See also

BindingSourceElement (Page 1943)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

BindingSourceElement.ReadCommand

Description

The "ReadCommand" property returns information about the command.

Type

Object, HmiCommandPart

Access

Read-only

Syntax

```
BindingSourceElement.ReadCommand
```

See also

BindingSourceElement (Page 1943)

Command.CommandParameters

Description

The "CommandParameters" property returns the parameters of the command.

Type

Object, HmiParameterCollection (Page 1953)

Access

Read-only

Syntax

```
Command.CommandParameters
```

See also

BindingSourceElement.ReadCommand (Page 1950)

HmiParameterCollection (Page 1953)

Command.CommandText

Description

The "CommandText" property returns the text of the command.

Type

String

Access

Read-only

Syntax

```
Command.CommandText
```

See also

BindingSourceElement.ReadCommand (Page 1950)

Command.CommandType

Description

The "CommandType" property returns the type of the command.

Type

String

Access

Read-only

Syntax

```
Command.CommandType
```

See also

BindingSourceElement.ReadCommand (Page 1950)

Command.Execute()

Description

The "Execute" method executes the command.

Syntax

```
Command.Execute()
```

Parameters

--

Return value

--

See also

BindingSourceElement.ReadCommand (Page 1950)

HmiParameterCollection

Description

The "HmiParameterCollection" object is a list of all parameters ("Parameter" objects).

Use

The "HmiParameterCollection" object is a list which can be counted and enumerated. You can access the "HmiParameterCollection" list using the index or the tag names.

Object type

HmiParameterCollection

Properties

The "HmiParameterCollection" object has the following properties:

- **Count**
  Returns the number of parameters in the "HmiParameterCollection" list.

Methods

The "HmiParameterCollection" object has the following methods:

- **Item()**
  Returns a parameter of the "HmiParameterCollection" list.

HmiParameterCollection.Count

Description

The "Count" property returns the number of parameters in the "HmiParameterCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiParameterCollection.Count
```

See also

HmiParameterCollection (Page 1953)

HmiParameterCollection.Item()

Description

The "Item" method returns a parameter of the "HmiParameterCollection" list.

Syntax

```
HmiParameterCollection.[.Item](HmiParameterName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiParameterCollection" object.

Parameters

`HmiParameterName`

Type: String

Name of the parameter

Return value

Object, HmiParameterPart (Page 1955)

See also

Parameter (Page 1955)

HmiParameterCollection (Page 1953)

Parameter

Description

The "Parameter" object represents a parameter.

Object type

HmiParameterPart

Properties

The "Parameter" object has the following properties:

- **ParameterName**
  Specifies the name of the parameter.
- **ParameterValue**
  Specifies the value of the parameter.

Methods

--

See also

HmiParameterCollection (Page 1953)

Parameter.ParameterName

Description

The "ParameterName" property specifies the name of the parameter.

Type

String

Access

Read-write

Syntax

```
Parameter.ParameterName
```

See also

Parameter (Page 1955)

Parameter.ParameterValue

Description

The "ParameterValue" property specifies the value of the parameter.

Type

Variant

Access

Read-write

Syntax

```
Parameter.ParameterValue
```

See also

Parameter (Page 1955)

BindingSourceElement.RowCount

Description

The "RowCount" property returns the number of rows.

Type

UInt32

Access

Read-only

Syntax

```
BindingSourceElement.RowCount
```

See also

BindingSourceElement (Page 1943)

BindingSourceElement.SourceState

Description

The "SourceState" property returns the state of the data source.

Type

Int32, HmiSourceState

Returns the state:

- Idle (0): Ready
- Busy (1): Busy

Access

Read-only

Syntax

```
BindingSourceElement.SourceState
```

See also

BindingSourceElement (Page 1943)

BindingSourceElement.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the data source.

Syntax

```
BindingSourceElement.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

BindingSourceElement (Page 1943)

BindingSourceElement.MoveFirst()

Description

The "MoveFirst" method moves the data source to the first position.

Syntax

```
BindingSourceElement.MoveFirst()
```

Parameters

--

Return value

--

See also

BindingSourceElement (Page 1943)

BindingSourceElement.MoveLast()

Description

The "MoveLast" method moves the data source to the last position.

Syntax

```
BindingSourceElement.MoveLast()
```

Parameters

--

Return value

See also

BindingSourceElement (Page 1943)

BindingSourceElement.MoveNext()

Description

The "MoveNext" method moves the data source forward.

Syntax

```
BindingSourceElement.MoveNext()
```

Parameters

--

Return value

See also

BindingSourceElement (Page 1943)

BindingSourceElement.MovePrevious()

Description

The "MovePrevious" method moves the data source backwards.

Syntax

```
BindingSourceElement.MovePrevious()
```

Parameters

--

Return value

--

See also

BindingSourceElement (Page 1943)

BindingSourceElement.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
BindingSourceElement.PropertyFlashing(propertyName, enable[, value]
[, alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

BindingSourceElement (Page 1943)

Button

Description

The "Button" object represents a button.

You can reference a "Button" object using the "ScreenItems" list or the "FindItem" method.

Object type

HmiButton

Properties

The "Button" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.

- **Content**
  Specifies display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags that affect the button.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Layer**
  Returns the screen layer in which the button is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the button.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the button was created.
- **RequireExplicitUnlock**
  Returns whether the object can only be operated while the associated button is being pressed.
- **RotationAngle**
  Specifies the rotation angle in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the button rotates.

- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies that the focus rectangle is displayed.
- **StyleItemClass**
  Returns the style which is applied to the button.
- **TabIndex**
  Returns the position of the button in the tab sequence.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the button is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "Button" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the button.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Button" object has the following events:

- **OnActivated()**
  Occurs when a button receives focus.
- **OnContextTapped()**
  Occurs when a button is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a button loses focus.
- **OnDown()**
  Occurs when the operator presses a button.

- **OnKeyDown()**
  Occurs when a button is pressed while the button is in focus.
- **OnKeyUp()**
  Occurs when a button is released while the button is in focus.
- **OnTapped()**
  Occurs when a button is left-clicked or short-touched.
- **OnUp()**
  Occurs when the operator resolves the pressure on a button via the input device.

Button.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Button.AlternateBackColor
```

See also

Button (Page 1961)

Button.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Button.AlternateBorderColor
```

See also

Button (Page 1961)

Button.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the button for the "pressed" state.

Type

String

Access

Read-write

Syntax

```
Button.AlternateGraphic
```

See also

Button (Page 1961)

Button.AlternateText

Description

The "AlternateText" property specifies the text of the button for the "pressed" state.

Type

String

Access

Read-write

Syntax

```
Button.AlternateText
```

See also

Button (Page 1961)

Button.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Button.Authorization
```

See also

Button (Page 1961)

Button.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Button.BackColor
```

See also

Button (Page 1961)

Button.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Button.BorderColor
```

See also

Button (Page 1961)

Button.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Button.BorderWidth
```

See also

Button (Page 1961)

Button.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
Button.Content
```

See also

Button (Page 1961)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

Button.Content (Page 1968)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

Button.Content (Page 1968)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

Button.Content (Page 1968)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

Button.Content (Page 1968)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

Button.Content (Page 1968)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

Button.Content (Page 1968)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

Button.Content (Page 1968)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

Button.Content (Page 1968)

Button.CurrentQuality

Description

The "CurrentQuality" property returns the current worst quality code of all tags which influence the button.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable. Quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Button.CurrentQuality
```

See also

Button (Page 1961)

Button.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Button.Enabled
```

See also

Button (Page 1961)

Button.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Button.Font
```

See also

Button (Page 1961)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Button.Font (Page 1974)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Button.Font (Page 1974)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Button.Font (Page 1974)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Button.Font (Page 1974)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Button.Font (Page 1974)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Button.Font (Page 1974)

Button.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Button.ForeColor
```

See also

Button (Page 1961)

Button.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
Button.Graphic
```

See also

Button (Page 1961)

Button.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

Button (Page 1961)

Button.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Button.Height
```

See also

Button (Page 1961)

Button.Layer

Description

The "Layer" property returns the layer of the screen in which the button is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Button.Layer
```

See also

Button (Page 1961)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Button.Layer (Page 1980)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Button.Layer (Page 1980)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Button.Layer (Page 1980)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Button.Layer (Page 1980)

Button.Left

Description

The "Left" property specifies the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Button.Left
```

See also

Button (Page 1961)

Button.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Button.Margin
```

See also

Button (Page 1961)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Button.Margin (Page 1983)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Button.Margin (Page 1983)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Button.Margin (Page 1983)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Button.Margin (Page 1983)

Button.Name

Description

The "Name" property returns the name of the button.

Type

String

Access

Read-only

Syntax

```
Button.Name
```

See also

Button (Page 1961)

Button.Opacity

Description

The "Opacity" property specifies the opacity of the button. The value "0.0" means completely transparent.

Type

Float

Access

Read-write

Syntax

```
Button.Opacity
```

See also

Button (Page 1961)

Button.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Button.Operability
```

See also

Button (Page 1961)

Button.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
Button.Padding
```

See also

Button (Page 1961)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

Button.Padding (Page 1987)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

Button.Padding (Page 1987)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

Button.Padding (Page 1987)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

Button.Padding (Page 1987)

Button.Parent

Description

The "Parent" property returns the parent screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Button.Parent
```

See also

Button (Page 1961)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Button.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the button was created.

Type

String

Access

Read-only

Syntax

```
Button.RenderingTemplate
```

See also

Button (Page 1961)

Button.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the object can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Button.RequireExplicitUnlock
```

See also

Button (Page 1961)

Button.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle of the button in degrees.

Type

Int16

Access

Read-write

Syntax

```
Button.RotationAngle
```

See also

Button (Page 1961)

Button.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the button rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also be outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Button.RotationCenterPlacement
```

See also

Button (Page 1961)

Button.RotationCenterX (Page 1993)

Button.RotationCenterY (Page 1993)

Button.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point.

Type

Float

Access

Read-write

Syntax

```
Button.RotationCenterX
```

See also

Button (Page 1961)
Button.RotationCenterPlacement (Page 1992)
Button.RotationCenterY (Page 1993)

Button.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point.

Type

Float

Access

Read-write

Syntax

```
Button.RotationCenterY
```

See also

Button (Page 1961)
Button.RotationCenterX (Page 1993)
Button.RotationCenterPlacement (Page 1992)

Button.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the button is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Button.ShowFocusVisual
```

See also

Button (Page 1961)

Button.StyleItemClass

Description

The "StyleItemClass" property returns the style that will be applied to the button.

Type

String

Access

Read-only

Syntax

```
Button.StyleItemClass
```

See also

Button (Page 1961)

Button.TabIndex

Description

The "TabIndex" property returns the position of the button in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Button.TabIndex
```

See also

Button (Page 1961)

Button.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
Button.Text
```

See also

Button (Page 1961)

Button.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
Button.ToolTipText
```

See also

Button (Page 1961)

Button.Top

Description

The "Top" property specifies the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Button.Top
```

See also

Button (Page 1961)

Button.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
Button.Visible
```

See also

Button (Page 1961)

Button.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
Button.VisualizeQuality
```

See also

Button (Page 1961)

Button.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Button.Width
```

See also

Button (Page 1961)

Button.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the button.

Syntax

```
Button.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Button (Page 1961)

Button.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing means the change between two values of a property.

Syntax

```
Button.PropertyFlashing(propertyName,enable[,value][,alternateValue]
[,rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flashing frequency:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Button (Page 1961)

Button_OnActivated()

Description

The "OnActivated" event occurs when a button receives focus:

- A button is selected via the configured tab sequence.
- A button that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and receives focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Button_OnActivated(item)
```

Context

Item

Type: Object

Button at which the event occurs.

See also

Button (Page 1961)

Button_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A button is clicked with the right mouse button.
- A button is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnDown
2. OnUp
3. OnContextTapped

Syntax

```
Button_OnContextTapped(item,x,y,modifiers,trigger)
```

Context

Item

Type: Object

Button at which the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Button (Page 1961)

Button_OnDeactivated()

Description

The "OnDeactivated" event occurs when a button loses focus when the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Button_OnDeactivated(item)
```

Context

`Item`

Type: Object

Button at which the event occurs.

See also

Button (Page 1961)

Button_OnDown()

Description

The "OnDown" event occurs when the operator presses a button:

- A button is clicked with a mouse button.
- The <RETURN> or <SPACE> key is pressed when a button has the focus.
- A button is touched.

Order

The events are triggered in the following order:

1. OnDown
2. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
3. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
4. OnUp

Syntax

```
Button_OnDown(item,x,y,modifiers,trigger)
```

Context

`Item`

Type: Object

Button at which the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Button (Page 1961)

Button_OnKeyDown()

Description

The "OnKeyDown" event occurs when a button is pressed while the button is in focus. If the key is <RETURN> or <SPACE>, an "OnKeyDown" event is triggered before an "OnDown" event.

Syntax

```
Button_OnKeyDown(item,keyCode,modifiers)
```

Order

The events are triggered in the following order:

1. OnDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyDown
3. OnKeyUp
4. OnUp (if triggered by the <RETURN> or <SPACE> key)

Context

`Item`

Type: Object

Button at which the event occurs.

`keyCode`

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Button (Page 1961)

Button_OnKeyUp()

Description

The "OnKeyUp" event occurs when a button is released while the button is in focus. If the key is <RETURN> or <SPACE>, an event "OnUp" is triggered after the event "OnKeyUp".

Syntax

```
Button_OnKeyUp(item,keyCode,modifiers)
```

Order

The events are triggered in the following order:

1. OnDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyDown
3. OnKeyUp
4. OnUp (if triggered by the <RETURN> or <SPACE> key)

Context

Item

Type: Object

Button at which the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Button (Page 1961)

Button_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A button is clicked with the left mouse button.
- The <RETURN> or <SPACE> key is pressed when a button has the focus.
- A button is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnDown
2. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
3. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
4. OnUp
5. OnTapped

Syntax

```
Button_OnTapped(item,x,y,modifiers,trigger)
```

Context

`Item`

Type: Object

Button at which the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Button (Page 1961)

Button_OnUp()

Description

The event "OnUp" occurs when the operator releases the pressure on a button via the input device:

- The mouse button is released via a button.
- The <RETURN> or <SPACE> key is released when a button has the focus.
- The touch of a button is canceled.
- A button is exited while pressed.

This event does not occur as long as the operator keeps the button pressed.

Order

The events are triggered in the following order:

1. OnDown
2. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
3. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
4. OnUp

Syntax

```
Button_OnUp(item,x,y,modifiers,trigger)
```

Context

Item

Type: Object

Button at which the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Button (Page 1961)

CheckBoxGroup

Description

The "CheckBoxGroup" object represents a checkbox (group of checkboxes) in runtime.

Object type

HmiCheckBoxGroup

Properties

The "CheckBoxGroup" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **Content**
  Specifies the display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the checkbox.
- **Enabled**
  Specifies whether the checkbox can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the checkbox is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the checkbox.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the checkbox is operable.
- **Padding**
  Specifies the distance of the content from the border of the checkbox.
- **Parent**
  Returns the higher-level screen object.

- **ProcessValue**
  Specifies the process value.
- **RenderingTemplate**
  Returns the name of the template from which the check box was created.
- **RequireExplicitUnlock**
  Returns whether the checkbox is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the checkbox rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **SelectionItemHeight**
  Specifies the height of the checkbox entries.
- **SelectionItems**
  Returns the list of all the checkbox entries.
- **SelectorPosition**
  Specifies the horizontal alignment of the checkbox entries.
- **ShowFocusVisual**
  Specifies whether the checkbox is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the checkbox.
- **TabIndex**
  Returns the position of the checkbox in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the checkbox is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "CheckBoxGroup" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the checkbox.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "CheckBoxGroup" object has the following events:

- **OnActivated()**
  Occurs when a checkbox receives focus.
- **OnContextTapped()**
  Occurs when a checkbox is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a checkbox loses focus.
- **OnKeyDown()**
  Occurs when a button is pressed while the checkbox is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the checkbox is in focus.
- **OnTapped()**
  Occurs when a checkbox is left-clicked or short-touched.

CheckBoxGroup.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.AlternateBackColor
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.AlternateBorderColor
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
CheckBoxGroup.Authorization
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.BackColor
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.BorderColor
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
CheckBoxGroup.BorderWidth
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
CheckBoxGroup.Content
```

See also

CheckBoxGroup (Page 2010)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

CheckBoxGroup.Content (Page 2016)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

CheckBoxGroup.Content (Page 2016)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

CheckBoxGroup.Content (Page 2016)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

CheckBoxGroup.Content (Page 2016)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

CheckBoxGroup.Content (Page 2016)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

CheckBoxGroup.Content (Page 2016)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

CheckBoxGroup.Content (Page 2016)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

CheckBoxGroup.Content (Page 2016)

CheckBoxGroup.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the checkbox.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
CheckBoxGroup.CurrentQuality
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Enabled

Description

The "Enabled" property specifies whether the checkbox can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
CheckBoxGroup.Enabled
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
CheckBoxGroup.Font
```

See also

CheckBoxGroup (Page 2010)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

CheckBoxGroup.Font (Page 2023)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

CheckBoxGroup.Font (Page 2023)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

CheckBoxGroup.Font (Page 2023)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

CheckBoxGroup.Font (Page 2023)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

CheckBoxGroup.Font (Page 2023)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

CheckBoxGroup.Font (Page 2023)

CheckBoxGroup.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.ForeColor
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.Height
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Layer

Description

The "Layer" property returns the layer of the screen in which the checkbox is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
CheckBoxGroup.Layer
```

See also

CheckBoxGroup (Page 2010)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

CheckBoxGroup.Layer (Page 2027)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

CheckBoxGroup.Layer (Page 2027)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

CheckBoxGroup.Layer (Page 2027)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

CheckBoxGroup.Layer (Page 2027)

CheckBoxGroup.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
CheckBoxGroup.Left
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
CheckBoxGroup.Margin
```

See also

CheckBoxGroup (Page 2010)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

CheckBoxGroup.Margin (Page 2030)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

CheckBoxGroup.Margin (Page 2030)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

CheckBoxGroup.Margin (Page 2030)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

CheckBoxGroup.Margin (Page 2030)

CheckBoxGroup.Name

Description

The "Name" property returns the name of the checkbox.

Type

String

Access

Read-only

Syntax

```
CheckBoxGroup.Name
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
CheckBoxGroup.Opacity
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Operability

Description

The "Operability" property returns whether the checkbox is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
CheckBoxGroup.Operability
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the checkbox.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
CheckBoxGroup.Padding
```

See also

CheckBoxGroup (Page 2010)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

CheckBoxGroup.Padding (Page 2034)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

CheckBoxGroup.Padding (Page 2034)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

CheckBoxGroup.Padding (Page 2034)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

CheckBoxGroup.Padding (Page 2034)

CheckBoxGroup.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
CheckBoxGroup.Parent
```

See also

CheckBoxGroup (Page 2010)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

CheckBoxGroup.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
CheckBoxGroup.ProcessValue
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the checkbox was created.

Type

String

Access

Read-only

Syntax

```
CheckBoxGroup.RenderingTemplate
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the checkbox can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
CheckBoxGroup.RequireExplicitUnlock
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
CheckBoxGroup.RotationAngle
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RotationCenterPlacement (Page 2040)

CheckBoxGroup.RotationCenterX (Page 2040)

CheckBoxGroup.RotationCenterY (Page 2041)

CheckBoxGroup.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the checkbox rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
CheckBoxGroup.RotationCenterPlacement
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RotationAngle (Page 2039)

CheckBoxGroup.RotationCenterX (Page 2040)

CheckBoxGroup.RotationCenterY (Page 2041)

CheckBoxGroup.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CheckBoxGroup.RotationCenterX
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RotationAngle (Page 2039)

CheckBoxGroup.RotationCenterPlacement (Page 2040)

CheckBoxGroup.RotationCenterY (Page 2041)

CheckBoxGroup.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CheckBoxGroup.RotationCenterY
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.RotationAngle (Page 2039)

CheckBoxGroup.RotationCenterPlacement (Page 2040)

CheckBoxGroup.RotationCenterX (Page 2040)

CheckBoxGroup.SelectionItemHeight

Description

The "SelectionItemHeight" property specifies the height of the checkbox entries. The value "0" indicates that the height is calculated automatically.

Type

UInt16

Access

Read-write

Syntax

```
CheckBoxGroup.SelectionItemHeight
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.SelectionItems

Description

The "SelectionItems" property returns the list of all checkbox entries ("SelectionItem" objects).

Type

Object, HmiSelectionItemCollection (Page 2043)

Access

Read-only

Syntax

```
CheckBoxGroup.SelectionItems
```

See also

CheckBoxGroup (Page 2010)

HmiSelectionItemCollection (Page 2043)

HmiSelectionItemCollection

Description

The "HmiSelectionItemCollection" object is a list of all entries ("SelectionItem" objects) of a list object.

Use

The "HmiSelectionItemCollection" object is a list and can be counted and enumerated. You can access the "HmiSelectionItemCollection" list using the index or the tag name.

Object type

HmiSelectionItemCollection

Properties

The "HmiSelectionItemCollection" object has the following properties:

- **Count**
  Returns the number of list entries of the "HmiSelectionItemCollection" list.

Methods

The "HmiSelectionItemCollection" object has the following methods:

- **Item()**
  Returns a list entry of the "HmiSelectionItemCollection" list.

See also

CheckBoxGroup.SelectionItems (Page 2042)

HmiSelectionItemCollection.Count

Description

The "Count" property returns the number of list entries in the "HmiSelectionItemCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiSelectionItemCollection.Count
```

See also

HmiSelectionItemCollection (Page 2043)

HmiSelectionItemCollection.Item()

Description

The "Item" method returns a list entry of the "HmiSelectionItemCollection" list.

Syntax

```
HmiSelectionItemCollection[.Item](HmiSelectionItemName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiSelectionItemCollection" object.

Parameters

`HmiSelectionItemName`

Type: String

Name of the list entry

Return value

Object, HmiSelectionItemPart (Page 2044)

See also

HmiSelectionItemCollection (Page 2043)

SelectionItem (Page 2044)

SelectionItem

Description

The "SelectionItem" object represents a list entry.

Object type

HmiSelectionItemPart

Properties

The "SelectionItem" object has the following properties:

- **Graphic**
  Specifies the graphic of the list entry.
- **IsSelected**
  Specifies whether the list entry is selected.
- **Text**
  Specifies the list entry text.

Methods

--

See also

HmiSelectionItemCollection (Page 2043)

SelectionItem.Graphic

Description

The "Graphic" property specifies the graphic of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Graphic
```

See also

SelectionItem (Page 2044)

SelectionItem.IsSelected

Description

The "IsSelected" property specifies whether the list entry is selected.

Type

Bool

Access

Read-write

Syntax

```
SelectionItem.IsSelected
```

See also

SelectionItem (Page 2044)

SelectionItem.Text

Description

The "Text" property specifies the text of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Text
```

See also

SelectionItem (Page 2044)

CheckBoxGroup.SelectorPosition

Description

The "SelectorPosition" property specifies the horizontal alignment of the checkbox entries.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
CheckBoxGroup.SelectorPosition
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the checkbox is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
CheckBoxGroup.ShowFocusVisual
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the checkbox.

Type

String

Access

Read-only

Syntax

```
CheckBoxGroup.StyleItemClass
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.TabIndex

Description

The "TabIndex" property returns the position of the checkbox in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
CheckBoxGroup.TabIndex
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
CheckBoxGroup.ToolTipText
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
CheckBoxGroup.Top
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Visible

Description

The "Visible" property specifies whether the checkbox is visible.

Type

Bool

Access

Read-write

Syntax

```
CheckBoxGroup.Visible
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
CheckBoxGroup.VisualizeQuality
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
CheckBoxGroup.Width
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the checkbox.

Syntax

```
CheckBoxGroup.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
CheckBoxGroup.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup_OnActivated()

Description

The "OnActivated" event occurs when a checkbox receives focus:

- A checkbox is selected via the configured tab sequence.
- A checkbox that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
CheckBoxGroup_OnActivated(item)
```

Context

item

Type: Object

Checkbox where the event occurs.

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A checkbox is right-clicked.
- A checkbox is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
CheckBoxGroup_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Checkbox where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup_OnDeactivated()

Description

The "OnDeactivated" event occurs when the checkbox loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
CheckBoxGroup_OnDeactivated(item)
```

Context

`item`

Type: Object

Checkbox where the event occurs.

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the checkbox is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CheckBoxGroup_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Checkbox where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the checkbox is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CheckBoxGroup_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Checkbox where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CheckBoxGroup (Page 2010)

CheckBoxGroup_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A checkbox is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a checkbox has the focus.
- A checkbox is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
CheckBoxGroup_OnTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Checkbox where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CheckBoxGroup (Page 2010)

Circle

Description

The "Circle" object represents a circle in runtime.

Object type

HmiCircle

Properties

The "Circle" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CenterX**
  Specifies the X coordinate of the center.
- **CenterY**
  Sets the Y coordinate of the center.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the circle.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the circle can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the circle is filled.
- **FillLevel**
  Specifies the fill of the circle in percent.
- **Layer**
  Returns the layer of the screen in which the circle is located.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the circle.
- **Opacity**
  Specifies the opacity.

- **Operability**
  Returns whether the circle is operable.
- **Parent**
  Returns the higher-level screen object.
- **Radius**
  Specifies the radius.
- **RequireExplicitUnlock**
  Returns whether the circle is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the circle rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the circle is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the circle.
- **TabIndex**
  Returns the position of the circle in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the circle is visible.

Methods

The "Circle" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the circle.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Circle" object has the following events:

- **OnActivated()**
  Occurs when a circle receives focus.
- **OnContextTapped()**
  Occurs when a circle is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a circle loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the circle is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the circle is in focus.
- **OnTapped()**
  Occurs when a circle is left-clicked or short-touched.

Circle.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Circle.AlternateBackColor
```

See also

Circle (Page 2059)

Circle.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Circle.AlternateBorderColor
```

See also

Circle (Page 2059)

Circle.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Circle.Authorization
```

See also

Circle (Page 2059)

Circle.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Circle.BackColor
```

See also

Circle (Page 2059)

Circle.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonally striped
- BackwardDiagonal (131075): Backward diagonally striped
- Cross (131076): Grid
- DiagonalCross (131077): Diagonal grid
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Gradient vertical tricolor

- GradientForwardDiagonalTricolor (1048834): Gradient forward diagonal tricolor
- GradientBackwardDiagonalTricolor (1048835): Gradient backward diagonal tricolor

Access

Read-write

Syntax

```
Circle.BackFillPattern
```

See also

Circle (Page 2059)
Circle.FillDirection (Page 2069)
Circle.FillLevel (Page 2070)
Circle.ShowFillLevel (Page 2081)

Circle.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Circle.BorderColor
```

See also

Circle (Page 2059)

Circle.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Circle.BorderWidth
```

See also

Circle (Page 2059)

Circle.CenterX

Description

The "CenterX" property specifies the X coordinate of the center.

Type

Int32

Access

Read-write

Syntax

```
Circle.CenterX
```

See also

Circle (Page 2059)

Circle.CenterY

Description

The "CenterY" property specifies the Y coordinate of the center.

Type

Int32

Access

Read-write

Syntax

```
Circle.CenterY
```

See also

Circle (Page 2059)

Circle.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the circle.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Circle.CurrentQuality
```

See also

Circle (Page 2059)

Circle.DashType

Description

The "DashType" property specifies the dash type of the border or the line.

Type

Int32, HmiDashType

Specifies the dash type:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dot
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Circle.DashType
```

See also

Circle (Page 2059)

Circle.Enabled

Description

The "Enabled" property specifies whether the circle can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Circle.Enabled
```

See also

Circle (Page 2059)

Circle.FillDirection

Description

The "FillDirection" property specifies the direction from which the circle is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom
- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
Circle.FillDirection
```

See also

Circle (Page 2059)

Circle.BackFillPattern (Page 2064)

Circle.FillLevel (Page 2070)
Circle.ShowFillLevel (Page 2081)

Circle.FillLevel

Description

The "FillLevel" property specifies the fill level of the circle in percent.

Type

UInt8

Access

Read-write

Syntax

```
Circle.FillLevel
```

See also

Circle (Page 2059)
Circle.BackFillPattern (Page 2064)
Circle.FillDirection (Page 2069)
Circle.ShowFillLevel (Page 2081)

Circle.Layer

Description

The "Layer" property returns the layer of the screen in which the circle is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Circle.Layer
```

See also

Circle (Page 2059)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Circle.Layer (Page 2070)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Circle.Layer (Page 2070)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Circle.Layer (Page 2070)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Circle.Layer (Page 2070)

Circle.Margin

Description

The Margin property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Circle.Margin
```

See also

Circle (Page 2059)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Circle.Margin (Page 2073)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Circle.Margin (Page 2073)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Circle.Margin (Page 2073)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Circle.Margin (Page 2073)

Circle.Name

Description

The "Name" property returns the name of the circle.

Type

String

Access

Read-only

Syntax

```
Circle.Name
```

See also

Circle (Page 2059)

Circle.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" means completely transparent.

Type

Float

Access

Read-write

Syntax

```
Cricle.Opacity
```

See also

Circle (Page 2059)

Circle.Operability

Description

The "Operability" property returns whether the circle is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Circle.Operability
```

See also

Circle (Page 2059)

Circle.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Circle.Parent
```

See also

Circle (Page 2059)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Circle.Radius

Description

The "Radius" property specifies the radius.

Type

UInt32

Access

Read-write

Syntax

```
Circle.Radius
```

See also

Circle (Page 2059)

Circle.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the circle can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Circle.RequireExplicitUnlock
```

See also

Circle (Page 2059)

Circle.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Circle.RotationAngle
```

See also

Circle (Page 2059)

Circle.RotationCenterPlacement (Page 2079)

Circle.RotationCenterX (Page 2080)

Circle.RotationCenterY (Page 2081)

Circle.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the circle rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also be outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Circle.RotationCenterPlacement
```

See also

Circle (Page 2059)

Circle.RotationAngle (Page 2079)

Circle.RotationCenterX (Page 2080)

Circle.RotationCenterY (Page 2081)

Circle.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Circle.RotationCenterX
```

See also

Circle (Page 2059)

Circle.RotationAngle (Page 2079)

Circle.RotationCenterPlacement (Page 2079)

Circle.RotationCenterY (Page 2081)

Circle.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Circle.RotationCenterY
```

See also

Circle (Page 2059)

Circle.RotationAngle (Page 2079)

Circle.RotationCenterPlacement (Page 2079)

Circle.RotationCenterX (Page 2080)

Circle.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
Circle.ShowFillLevel
```

See also

Circle (Page 2059)
Circle.BackFillPattern (Page 2064)
Circle.FillDirection (Page 2069)
Circle.FillLevel (Page 2070)

Circle.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the circle is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Circle.ShowFocusVisual
```

See also

Circle (Page 2059)

Circle.StyleItemClass

Description

The "StyleItemClass" property returns the style applied to the circle.

Type

String

Access

Read-only

Syntax

```
Circle.StyleItemClass
```

See also

Circle (Page 2059)

Circle.TabIndex

Description

The "TabIndex" property returns the position of the circle in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Circle.TabIndex
```

See also

Circle (Page 2059)

Circle.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Circle.ToolTipText
```

See also

Circle (Page 2059)

Circle.Visible

Description

The "Visible" property specifies whether the circle is visible.

Type

Bool

Access

Read-write

Syntax

```
Circle.Visible
```

See also

Circle (Page 2059)

Circle.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the circle.

Syntax

```
Circle.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Circle (Page 2059)

Circle.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Circle.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Circle (Page 2059)

Circle_OnActivated()

Description

The "OnActivated" event occurs when a circle receives focus:

- A circle is selected via the configured tab sequence.
- A circle that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and receives focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Circle_OnActivated(item)
```

Context

item

Type: Object

Circle at which the event occurs.

See also

Circle (Page 2059)

Circle_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A circle is clicked with the right mouse button.
- A circle is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Circle_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Circle where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Circle (Page 2059)

Circle_OnDeactivated()

Description

The "OnDeactivated" event occurs when a circle loses focus when the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Circle_OnDeactivated(item)
```

Context

`item`

Type: Object

Circle at which the event occurs.

See also

Circle (Page 2059)

Circle_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the circle is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Circle_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Circle where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Circle (Page 2059)

Circle_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the circle is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Circle_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Circle where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Circle (Page 2059)

Circle_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A circle is clicked with the left mouse button.
- The <RETURN> or <SPACE> key is pressed when a circle has the focus.
- A circle is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Circle_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Circle where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Circle (Page 2059)

CircleSegment

Description

The "CircleSegment" object represents a circle segment in runtime.

Object type

HmiCircleSegment

Properties

The "CircleSegment" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AngleRange**
  Specifies the arc angle.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CenterX**
  Specifies the X coordinate of the rotation point.
- **CenterY**
  Specifies the Y coordinate of the rotation point.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the circle segment.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the circle segment can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the circle segment is filled.
- **FillLevel**
  Specifies the fill of the circle segment in percent.
- **Layer**
  Returns the layer of the screen in which the circle segment is located.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the circle segment.

- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the circle segment is operable.
- **Parent**
  Returns the higher-level screen object.
- **Radius**
  Specifies the radius.
- **RequireExplicitUnlock**
  Returns whether the circle segment is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the circle segment rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the circle segment is highlighted when in focus.
- **StartAngle**
  Specifies the angle by which the start point deviates from the zero position (0°).
- **StyleItemClass**
  Returns the style which is applied to the circle segment.
- **TabIndex**
  Returns the position of the circle segment in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the circle segment is visible.

Methods

The "CircleSegment" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the circle segment.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "CircleSegment" object has the following events:

- **OnActivated()**
  Occurs when a circle segment receives focus.
- **OnContextTapped()**
  Occurs when a circle segment is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a circle segment loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the circle segment is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the circle segment is in focus.
- **OnTapped()**
  Occurs when a circle segment is left-clicked or short-touched.

CircleSegment.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
CircleSegment.AlternateBackColor
```

See also

CircleSegment (Page 2092)

CircleSegment.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
CircleSegment.AlternateBorderColor
```

See also

CircleSegment (Page 2092)

CircleSegment.AngleRange

Description

The AngleRange property specifies the arc angle clockwise.

Type

Int32

Access

Read-write

Syntax

```
CircleSegment.AngleRange
```

See also

CircleSegment (Page 2092)

CircleSegment.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
CircleSegment.Authorization
```

See also

CircleSegment (Page 2092)

CircleSegment.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
CircleSegment.BackColor
```

See also

CircleSegment (Page 2092)

CircleSegment.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent

- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonally striped
- BackwardDiagonal (131075): Backward diagonally striped
- Cross (131076): Grid
- DiagonalCross (131077): Diagonal grid
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Gradient vertical tricolor
- GradientForwardDiagonalTricolor (1048834): Gradient forward diagonal tricolor
- GradientBackwardDiagonalTricolor (1048835): Gradient backward diagonal tricolor

Access

Read-write

Syntax

```
CircleSegment.BackFillPattern
```

See also

CircleSegment (Page 2092)
CircleSegment.FillDirection (Page 2102)
CircleSegment.FillLevel (Page 2103)
CircleSegment.ShowFillLevel (Page 2114)

CircleSegment.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
CircleSegment.BorderColor
```

See also

CircleSegment (Page 2092)

CircleSegment.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
CircleSegment.BorderWidth
```

See also

CircleSegment (Page 2092)

CircleSegment.CenterX

Description

The "CenterX" property specifies the X coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
CircleSegment.CenterX
```

See also

CircleSegment (Page 2092)

CircleSegment.CenterY

Description

The "CenterY" property specifies the Y coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
CircleSegment.CenterY
```

See also

CircleSegment (Page 2092)

CircleSegment.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the circle segment.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.

- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
CircleSegment.CurrentQuality
```

See also

CircleSegment (Page 2092)

CircleSegment.DashType

Description

The "DashType" property specifies the dash type of the border or the line.

Type

Int32, HmiDashType

Specifies the dash type:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dot
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
CircleSegment.DashType
```

See also

CircleSegment (Page 2092)

CircleSegment.Enabled

Description

The "Enabled" property specifies whether the circle segment can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
CircleSegment.Enabled
```

See also

CircleSegment (Page 2092)

CircleSegment.FillDirection

Description

The "FillDirection" property specifies the direction from which the circle segment is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom
- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
CircleSegment.FillDirection
```

See also

CircleSegment (Page 2092)
CircleSegment.BackFillPattern (Page 2097)
CircleSegment.FillLevel (Page 2103)
CircleSegment.ShowFillLevel (Page 2114)

CircleSegment.FillLevel

Description

The "FillLevel" property specifies the fill level of the circle segment in percent.

Type

UInt8

Access

Read-write

Syntax

```
CircleSegment.FillLevel
```

See also

CircleSegment (Page 2092)
CircleSegment.BackFillPattern (Page 2097)
CircleSegment.FillDirection (Page 2102)
CircleSegment.ShowFillLevel (Page 2114)

CircleSegment.Layer

Description

The "Layer" property returns the layer of the screen in which the circle segment is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
CircleSegment.Layer
```

See also

CircleSegment (Page 2092)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

CircleSegment.Layer (Page 2103)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

CircleSegment.Layer (Page 2103)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

CircleSegment.Layer (Page 2103)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

CircleSegment.Layer (Page 2103)

CircleSegment.Margin

Description

The Margin property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
CircleSegment.Margin
```

See also

CircleSegment (Page 2092)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

CircleSegment.Margin (Page 2106)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

CircleSegment.Margin (Page 2106)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

CircleSegment.Margin (Page 2106)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

CircleSegment.Margin (Page 2106)

CircleSegment.Name

Description

The "Name" property returns the name of the circle segment.

Type

String

Access

Read-only

Syntax

```
CircleSegment.Name
```

See also

CircleSegment (Page 2092)

CircleSegment.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" means completely transparent.

Type

Float

Access

Read-write

Syntax

```
CircleSegment.Opacity
```

See also

CircleSegment (Page 2092)

CircleSegment.Operability

Description

The "Operability" property returns whether the circle segment is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.

- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
CircleSegment.Operability
```

See also

CircleSegment (Page 2092)

CircleSegment.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
CircleSegment.Parent
```

See also

CircleSegment (Page 2092)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

CircleSegment.Radius

Description

The "Radius" property specifies the radius.

Type

UInt32

Access

Read-write

Syntax

```
CircleSegment.Radius
```

See also

CircleSegment (Page 2092)

CircleSegment.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the circle segment can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
CircleSegment.RequireExplicitUnlock
```

See also

CircleSegment (Page 2092)

CircleSegment.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
CircleSegment.RotationAngle
```

See also

CircleSegment (Page 2092)

CircleSegment.RotationCenterPlacement (Page 2112)

CircleSegment.RotationCenterX (Page 2113)

CircleSegment.RotationCenterY (Page 2114)

CircleSegment.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the circle segment rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also be outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
CircleSegment.RotationCenterPlacement
```

See also

CircleSegment (Page 2092)

CircleSegment.RotationAngle (Page 2112)

CircleSegment.RotationCenterX (Page 2113)

CircleSegment.RotationCenterY (Page 2114)

CircleSegment.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CircleSegment.RotationCenterX
```

See also

CircleSegment (Page 2092)

CircleSegment.RotationAngle (Page 2112)

CircleSegment.RotationCenterPlacement (Page 2112)

CircleSegment.RotationCenterY (Page 2114)

CircleSegment.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CircleSegment.RotationCenterY
```

See also

CircleSegment (Page 2092)

CircleSegment.RotationAngle (Page 2112)

CircleSegment.RotationCenterPlacement (Page 2112)

CircleSegment.RotationCenterX (Page 2113)

CircleSegment.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
CircleSegment.ShowFillLevel
```

See also

CircleSegment (Page 2092)

CircleSegment.BackFillPattern (Page 2097)

CircleSegment.FillDirection (Page 2102)

CircleSegment.FillLevel (Page 2103)

CircleSegment.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the circle segment is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
CircleSegment.ShowFocusVisual
```

See also

CircleSegment (Page 2092)

CircleSegment.StartAngle

Description

The "StartAngle" specifies the angle by which the start point deviates from the zero position (0° corresponds to 3 o'clock).

Type

Int32

Access

Read-write

Syntax

```
CircleSegment.StartAngle
```

See also

CircleSegment (Page 2092)

CircleSegment.StyleItemClass

Description

The "StyleItemClass" property returns the style applied to the circle segment.

Type

String

Access

Read-only

Syntax

```
CircleSegment.StyleItemClass
```

See also

CircleSegment (Page 2092)

CircleSegment.TabIndex

Description

The "TabIndex" property returns the position of the circle segment in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
CircleSegment.TabIndex
```

See also

CircleSegment (Page 2092)

CircleSegment.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
CircleSegment.ToolTipText
```

See also

CircleSegment (Page 2092)

CircleSegment.Visible

Description

The "Visible" property specifies whether the circle segment is visible.

Type

Bool

Access

Read-write

Syntax

```
CircleSegment.Visible
```

See also

CircleSegment (Page 2092)

CircleSegment.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the circle segment.

Syntax

```
CircleSegment.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

CircleSegment (Page 2092)

CircleSegment.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
CircleSegment.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

CircleSegment (Page 2092)

CircleSegment_OnActivated()

Description

The "OnActivated" event occurs when a circle segment receives focus:

- A circle segment is selected via the configured tab sequence.
- A circle segment that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and receives focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
CircleSegment_OnActivated(item)
```

Context

`item`

Type: Object

Circle segment at which the event occurs.

See also

CircleSegment (Page 2092)

CircleSegment_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A circle segment is clicked with the right mouse button.
- A circle segment is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
CircleSegment_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Circle segment where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CircleSegment (Page 2092)

CircleSegment_OnDeactivated()

Description

The "OnDeactivated" event occurs when a circle segment loses focus when the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
CircleSegment_OnDeactivated(item)
```

Context

`item`

Type: Object

Circle segment at which the event occurs.

See also

CircleSegment (Page 2092)

CircleSegment_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the circle segment is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CircleSegment_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Circle segment where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CircleSegment (Page 2092)

CircleSegment_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the circle segment is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CircleSegment_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Circle segment where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CircleSegment (Page 2092)

CircleSegment_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A circle segment is clicked with the left mouse button.
- The <RETURN> or <SPACE> key is pressed when a circle segment has the focus.
- A circle segment is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
CircleSegment_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Circle segment where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CircleSegment (Page 2092)

CircularArc

Description

The "CircularArc" object represents a circular arc in runtime.

Object type

HmiCircularArc

Properties

The "CircularArc" object has the following properties:

- **AlternateLineColor**
  Specifies the second line color which is displayed for line styles such as "Dash".
- **AngleRange**
  Specifies the arc angle.
- **Authorization**
  Returns the operator authorization.
- **CapType**
  Specifies the shape of the line ends.
- **CenterX**
  Specifies the X coordinate of the rotation point.
- **CenterY**
  Specifies the Y coordinate of the rotation point.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the circular arc.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the circular arc can be operated in runtime.
- **EndType**
  Specifies the type of line end.
- **Layer**
  Returns the layer of the screen in which the circular arc is located.
- **LineColor**
  Specifies the line color.
- **LineWidth**
  Specifies the line thickness.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the circular arc.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the circular arc is operable.
- **Parent**
  Returns the higher-level screen object.
- **Radius**
  Specifies the radius.
- **RequireExplicitUnlock**
  Returns whether the circular arc is only operable while the associated button is being pressed.

- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the circular arc rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the circular arc is highlighted when in focus.
- **StartAngle**
  Specifies the angle by which the start point deviates from the zero position (0°).
- **StartType**
  Specifies the type of line start.
- **StyleItemClass**
  Returns the style which is applied to the circular arc.
- **TabIndex**
  Returns the position of the circular arc in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the circular arc is visible.

Methods

The "CircularArc" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the circular arc.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "CircularArc" object has the following events:

- **OnActivated()**
  Occurs when a circular arc receives focus.
- **OnContextTapped()**
  Occurs when a circular arc is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a circular arc loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the circular arc is in focus.

- **OnKeyUp()**
  Occurs when a key is released while the circular arc is in focus.
- **OnTapped()**
  Occurs when a circular arc is left-clicked or short-touched.

CircularArc.AlternateLineColor

Description

The "AlternateLineColor" property specifies the second line color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
CircularArc.AlternateLineColor
```

See also

CircularArc (Page 2126)

CircularArc.AngleRange

Description

The AngleRange property specifies the arc angle clockwise.

Type

Int32

Access

Read-write

Syntax

```
CircularArc.AngleRange
```

See also

CircularArc (Page 2126)

CircularArc.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
CircularArc.Authorization
```

See also

CircularArc (Page 2126)

CircularArc.CapType

Description

The "CapType" property specifies the shape of the line ends.

Type

Int32, HmiCapType

Specifies the line ends:

- Round (0): Round (line extends beyond the line end point with half the line thickness)
- Square (256): Square (line extends beyond the line end point with half the line thickness)
- Flat (512): Justified (line ends at the line end point)

Access

Read-write

Syntax

```
CircularArc.CapType
```

See also

CircularArc (Page 2126)

CircularArc.CenterX

Description

The "CenterX" property specifies the X coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
CircularArc.CenterX
```

See also

CircularArc (Page 2126)

CircularArc.CenterY

Description

The "CenterY" property specifies the Y coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
CircularArc.CenterY
```

See also

CircularArc (Page 2126)

CircularArc.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the circular arc.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
CircularArc.CurrentQuality
```

See also

CircularArc (Page 2126)

CircularArc.DashType

Description

The "DashType" property specifies the dash type of the border or the line.

Type

Int32, HmiDashType

Specifies the dash type:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dot
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
CircularArc.DashType
```

See also

CircularArc (Page 2126)

CircularArc.Enabled

Description

The "Enabled" property specifies whether the circular arc can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
CircularArc.Enabled
```

See also

CircularArc (Page 2126)

CircularArc.EndType

Description

The "EndType" property specifies the type of line end.

Type

Int32, HmiLineEndType

Specifies the end of the line:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reversed arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
CircularArc.EndType
```

See also

CircularArc (Page 2126)

CircularArc.Layer

Description

The "Layer" property returns the layer of the screen in which the circular arc is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
CircularArc.Layer
```

See also

CircularArc (Page 2126)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

CircularArc.Layer (Page 2134)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

CircularArc.Layer (Page 2134)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

CircularArc.Layer (Page 2134)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

CircularArc.Layer (Page 2134)

CircularArc.LineColor

Description

The "LineColor" property specifies the line color.

Type

UInt32

Access

Read-write

Syntax

```
CircularArc.LineColor
```

See also

CircularArc (Page 2126)

CircularArc.LineWidth

Description

The "LineWidth" property specifies the line thickness.

Type

UInt8

Access

Read-write

Syntax

```
CircularArc.LineWidth
```

See also

CircularArc (Page 2126)

CircularArc.Margin

Description

The Margin property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
CircularArc.Margin
```

See also

CircularArc (Page 2126)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

CircularArc.Margin (Page 2138)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

CircularArc.Margin (Page 2138)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

CircularArc.Margin (Page 2138)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

CircularArc.Margin (Page 2138)

CircularArc.Name

Description

The "Name" property returns the name of the circular arc.

Type

String

Access

Read-only

Syntax

```
CircularArc.Name
```

See also

CircularArc (Page 2126)

CircularArc.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" means completely transparent.

Type

Float

Access

Read-write

Syntax

```
CircularArc.Opacity
```

See also

CircularArc (Page 2126)

CircularArc.Operability

Description

The "Operability" property returns whether the circular arc is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
CircularArc.Operability
```

See also

CircularArc (Page 2126)

CircularArc.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
CircularArc.Parent
```

See also

CircularArc (Page 2126)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

CircularArc.Radius

Description

The "Radius" property specifies the radius.

Type

UInt32

Access

Read-write

Syntax

```
CircularArc.Radius
```

See also

CircularArc (Page 2126)

CircularArc.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the circular arc can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
CircularArc.RequireExplicitUnlock
```

See also

CircularArc (Page 2126)

CircularArc.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
CircularArc.RotationAngle
```

See also

CircularArc (Page 2126)

CircularArc.RotationCenterPlacement (Page 2144)

CircularArc.RotationCenterX (Page 2145)

CircularArc.RotationCenterY (Page 2145)

CircularArc.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the circular arc rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also be outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
CircularArc.RotationCenterPlacement
```

See also

CircularArc (Page 2126)

CircularArc.RotationAngle (Page 2143)

CircularArc.RotationCenterX (Page 2145)

CircularArc.RotationCenterY (Page 2145)

CircularArc.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CircularArc.RotationCenterX
```

See also

CircularArc (Page 2126)

CircularArc.RotationAngle (Page 2143)

CircularArc.RotationCenterPlacement (Page 2144)

CircularArc.RotationCenterY (Page 2145)

CircularArc.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CircularArc.RotationCenterY
```

See also

CircularArc (Page 2126)

CircularArc.RotationAngle (Page 2143)

CircularArc.RotationCenterX (Page 2145)

CircularArc.RotationCenterPlacement (Page 2144)

CircularArc.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the circular arc is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
CircularArc.ShowFocusVisual
```

See also

CircularArc (Page 2126)

CircularArc.StartAngle

Description

The "StartAngle" specifies the angle by which the start point deviates from the zero position (0° corresponds to 3 o'clock).

Type

Int32

Access

Read-write

Syntax

```
CircularArc.StartAngle
```

See also

CircularArc (Page 2126)

CircularArc.StartType

Description

The "StartType" property specifies the type of line start.

Type

Int32, HmiLineEndType

Specifies the start of the line:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reversed arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
CircularArc.StartType
```

See also

CircularArc (Page 2126)

CircularArc.StyleItemClass

Description

The "StyleItemClass" property returns the style applied to the circular arc.

Type

String

Access

Read-only

Syntax

```
CircularArc.StyleItemClass
```

See also

CircularArc (Page 2126)

CircularArc.TabIndex

Description

The "TabIndex" property returns the position of the circular arc in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
CircularArc.TabIndex
```

See also

CircularArc (Page 2126)

CircularArc.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
CircularArc.ToolTipText
```

See also

CircularArc (Page 2126)

CircularArc.Visible

Description

The "Visible" property specifies whether the circular arc is visible.

Type

Bool

Access

Read-write

Syntax

```
CircularArc.Visible
```

See also

CircularArc (Page 2126)

CircularArc.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the circular arc.

Syntax

```
CircularArc.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

CircularArc (Page 2126)

CircularArc.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
CircularArc.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

CircularArc (Page 2126)

CircularArc_OnActivated()

Description

The "OnActivated" event occurs when a circular arc receives focus:

- A circular arc is selected via the configured tab sequence.
- A circular arc that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and receives focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
CircularArc_OnActivated(item)
```

Context

`item`

Type: Object

Circle segment at which the event occurs.

See also

CircularArc (Page 2126)

CircularArc_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A circular arc is clicked with the right mouse button.
- A circular arc is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
CircularArc_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Circular arc where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CircularArc (Page 2126)

CircularArc_OnDeactivated()

Description

The "OnDeactivated" event occurs when an arc loses focus when the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
CircularArc_OnDeactivated(item)
```

Context

`item`

Type: Object

Circle segment at which the event occurs.

See also

CircularArc (Page 2126)

CircularArc_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the circular arc is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CircularArc_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Circular arc where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CircularArc (Page 2126)

CircularArc_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the circular arc is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CircularArc_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Circular arc where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CircularArc (Page 2126)

CircularArc_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A circular arc is clicked with the left mouse button.
- The <RETURN> or <SPACE> key is pressed when a circular arc has the focus.
- A circular arc is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
CircularArc_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Circular arc where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CircularArc (Page 2126)

Clock

Description

The "Clock" object represents a clock in runtime.

Object type

HmiClock

Properties

The "Clock" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the clock.
- **DialBackColor**
  Specifies the background color of the dial.
- **DialLabelColor**
  Specifies the color of the dial label.
- **DialLabelFont**
  Specifies the font of the dial.
- **DialMode**
  Specifies the details of the dial that are displayed.
- **DialTickColor**
  Specifies the color of the dial divisions.
- **Enabled**
  Specifies whether the clock can be operated in runtime.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the clock is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the clock.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the clock is operable.

- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the clock was created.
- **RequireExplicitUnlock**
  Returns whether the clock is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the clock rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the clock is highlighted when in focus.
- **ShowHours**
  Specifies whether the hour hand is displayed.
- **ShowMinutes**
  Specifies whether the minute hand is displayed.
- **ShowSeconds**
  Specifies whether the second hand is displayed.
- **StyleItemClass**
  Returns the style which is applied to the clock.
- **TabIndex**
  Returns the position of the clock in the tab sequence.
- **TimeSource**
  Specifies the source for the displayed time.
- **Title**
  Specifies the caption which appears as the title.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the clock is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "Clock" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the clock.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Clock" object has the following events:

- **OnActivated()**
  Occurs when a clock receives focus.
- **OnContextTapped()**
  Occurs when a clock is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a clock loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the clock is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the clock is in focus.
- **OnTapped()**
  Occurs when a clock is left-clicked or short-touched.

Clock.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Clock.AlternateBackColor
```

See also

Clock (Page 2158)

Clock.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Clock.AlternateBorderColor
```

See also

Clock (Page 2158)

Clock.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Clock.Authorization
```

See also

Clock (Page 2158)

Clock.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Clock.BackColor
```

See also

Clock (Page 2158)

Clock.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Clock.BorderColor
```

See also

Clock (Page 2158)

Clock.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Clock.BorderWidth
```

See also

Clock (Page 2158)

Clock.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the clock.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Clock.CurrentQuality
```

See also

Clock (Page 2158)

Clock.DialBackColor

Description

The "DialBackColor" property specifies the background color of the dial.

Type

UInt32

Access

Read-write

Syntax

```
Clock.DialBackColor
```

See also

Clock (Page 2158)

Clock.DialLabelColor

Description

The "DialLabelColor" property specifies the color of the dial labeling.

Type

UInt32

Access

Read-write

Syntax

```
Clock.DialLabelColor
```

See also

Clock (Page 2158)

Clock.DialLabelFont

Description

The "DialLabelFont" property specifies the font of the dial.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Clock.DialLabelFont
```

See also

Clock (Page 2158)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Clock.DialLabelFont (Page 2166)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Clock.DialLabelFont (Page 2166)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Clock.DialLabelFont (Page 2166)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Clock.DialLabelFont (Page 2166)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Clock.DialLabelFont (Page 2166)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Clock.DialLabelFont (Page 2166)

Clock.DialMode

Description

The "DialMode" property specifies the details of the dial that are displayed.

Type

Int32, HmiScaleMode

Specifies the displayed details:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
Clock.DialMode
```

See also

Clock (Page 2158)

Clock.DialTickColor

Description

The "DialTickColor" property specifies the color of the dial divisions.

Type

UInt32

Access

Read-write

Syntax

```
Clock.DialTickColor
```

See also

Clock (Page 2158)

Clock.Enabled

Description

The "Enabled" property specifies whether the parameter set display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Clock.Enabled
```

See also

Clock (Page 2158)

Clock.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Clock.Height
```

See also

Clock (Page 2158)

Clock.Layer

Description

The "Layer" property returns the layer of the screen in which the parameter set display is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Clock.Layer
```

See also

Clock (Page 2158)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Clock.Layer (Page 2172)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Clock.Layer (Page 2172)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Clock.Layer (Page 2172)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Clock.Layer (Page 2172)

Clock.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Clock.Left
```

See also

Clock (Page 2158)

Clock.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Clock.Margin
```

See also

Clock (Page 2158)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Clock.Margin (Page 2175)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Clock.Margin (Page 2175)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Clock.Margin (Page 2175)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Clock.Margin (Page 2175)

Clock.Name

Description

The "Name" property returns the name of the clock.

Type

String

Access

Read-only

Syntax

```
Clock.Name
```

See also

Clock (Page 2158)

Clock.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Clock.Opacity
```

See also

Clock (Page 2158)

Clock.Operability

Description

The "Operability" property returns whether the clock is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Clock.Operability
```

See also

Clock (Page 2158)

Clock.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Clock.Parent
```

See also

Clock (Page 2158)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Clock.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the clock was created.

Type

String

Access

Read-only

Syntax

```
Clock.RenderingTemplate
```

See also

Clock (Page 2158)

Clock.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the clock can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Clock.RequireExplicitUnlock
```

See also

Clock (Page 2158)

Clock.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Clock.RotationAngle
```

See also

Clock (Page 2158)

Clock.RotationCenterPlacement (Page 2181)

Clock.RotationCenterX (Page 2182)

Clock.RotationCenterY (Page 2182)

Clock.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the clock rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Clock.RotationCenterPlacement
```

See also

Clock (Page 2158)

Clock.RotationAngle (Page 2180)

Clock.RotationCenterX (Page 2182)

Clock.RotationCenterY (Page 2182)

Clock.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Clock.RotationCenterX
```

See also

Clock (Page 2158)

Clock.RotationAngle (Page 2180)

Clock.RotationCenterPlacement (Page 2181)

Clock.RotationCenterY (Page 2182)

Clock.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Clock.RotationCenterY
```

See also

Clock (Page 2158)
Clock.RotationAngle (Page 2180)
Clock.RotationCenterPlacement (Page 2181)
Clock.RotationCenterX (Page 2182)

Clock.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the clock is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Clock.ShowFocusVisual
```

See also

Clock (Page 2158)

Clock.ShowHours

Description

The "ShowHours" property specifies whether the hour hand is displayed.

Type

Bool

Access

Read-write

Syntax

```
Clock.ShowHours
```

See also

Clock (Page 2158)
Clock.ShowMinutes (Page 2184)
Clock.ShowSeconds (Page 2184)

Clock.ShowMinutes

Description

The "ShowMinutes" property specifies whether the minute hand is displayed.

Type

Bool

Access

Read-write

Syntax

```
Clock.ShowMinutes
```

See also

Clock (Page 2158)
Clock.ShowHours (Page 2183)
Clock.ShowSeconds (Page 2184)

Clock.ShowSeconds

Description

The "ShowSeconds" property specifies whether the second hand is displayed.

Type

Bool

Access

Read-write

Syntax

```
Clock.ShowSeconds
```

See also

Clock (Page 2158)
Clock.ShowHours (Page 2183)
Clock.ShowMinutes (Page 2184)

Clock.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the clock.

Type

String

Access

Read-only

Syntax

```
Clock.StyleItemClass
```

See also

Clock (Page 2158)

Clock.TabIndex

Description

The "TabIndex" property returns the position of the clock in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Clock.TabIndex
```

See also

Clock (Page 2158)

Clock.TimeSource

Description

The "TimeSource" property specifies the source for the displayed time. If the property is not configured, local time is used.

Type

Variant

Access

Read-write

Syntax

```
Clock.TimeSource
```

See also

Clock (Page 2158)

Clock.Title

Description

The "Title" property specifies the caption that appears as the title.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Clock.Title
```

See also

Clock (Page 2158)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Clock.Title (Page 2186)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2187)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2187)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2187)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2187)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2187)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2187)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Clock.Title (Page 2186)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Clock.Title (Page 2186)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Clock.Title (Page 2186)

Clock.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Clock.ToolTipText
```

See also

Clock (Page 2158)

Clock.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Clock.Top
```

See also

Clock (Page 2158)

Clock.Visible

Description

The "Visible" property specifies whether the clock is visible.

Type

Bool

Access

Read-write

Syntax

```
Clock.Visible
```

See also

Clock (Page 2158)

Clock.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
Clock.VisualizeQuality
```

See also

Clock (Page 2158)

Clock.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Clock.Width
```

See also

Clock (Page 2158)

Clock.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the clock.

Syntax

```
Clock.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Clock (Page 2158)

Clock.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Clock.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Clock (Page 2158)

Clock_OnActivated()

Description

The "OnActivated" event occurs when a clock receives focus:

- A clock is selected via the configured tab sequence.
- A clock that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Clock_OnActivated(item)
```

Context

item

Type: Object

Clock where the event occurs.

See also

Clock (Page 2158)

Clock_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A clock is right-clicked.
- A clock is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Clock_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Clock where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Clock (Page 2158)

Clock_OnDeactivated()

Description

The "OnDeactivated" event occurs when the clock loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Clock_OnDeactivated(item)
```

Context

item

Type: Object

Clock where the event occurs.

See also

Clock (Page 2158)

Clock_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the clock is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Clock_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Clock where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Clock (Page 2158)

Clock_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the clock is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Clock_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Clock where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Clock (Page 2158)

Clock_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A clock is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a clock has the focus.
- A clock is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Clock_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Clock where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Clock (Page 2158)

ComboBox

Description

The "ComboBox" object represents a combo box in runtime. The combo box is a combination of text box and list box.

Object type

HmiComboBox

Properties

The "ComboBox" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border width.
- **Content**
  Specifies display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the combo box.
- **Enabled**
  Specifies whether the combo box can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **Layer**
  Returns the screen layer in which the combo box is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the combo box.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the combo box is operable.
- **Padding**
  Specifies the distance of the content from the border of the combo box.
- **Parent**
  Returns the higher-level screen object.

- **ProcessValue**
  Specifies the process value.
- **RenderingTemplate**
  Returns the name of the template from which the combo box was created.
- **RequireExplicitUnlock**
  Returns whether the combo box is only operable while the corresponding button is being pressed.
- **RotationAngle**
  Specifies the rotation angle in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the combo box rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **SelectionItemHeight**
  Specifies the height of the list entries.
- **SelectionItems**
  Returns the list of all list entries in the combo box.
- **SelectionMode**
  Specifies whether one or more list entries can be selected in the combo box.
- **SelectorPosition**
  Specifies the horizontal alignment of the list entries.
- **ShowFocusVisual**
  Specifies whether the combo box is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the combo box.
- **TabIndex**
  Returns the position of the combo box in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the combo box is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "ComboBox" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the combo box.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ComboBox" object has the following events:

- **OnActivated()**
  Occurs when a combo box receives focus.
- **OnContextTapped()**
  Occurs when a combo box is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a combo box loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the combo box is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the combo box is in focus.
- **OnTapped()**
  Occurs when a combo box is left-clicked or short-touched.

ComboBox.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.AlternateBackColor
```

See also

ComboBox (Page 2203)

ComboBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.AlternateBorderColor
```

See also

ComboBox (Page 2203)

ComboBox.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ComboBox.Authorization
```

See also

ComboBox (Page 2203)

ComboBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.BackColor
```

See also

ComboBox (Page 2203)

ComboBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.BorderColor
```

See also

ComboBox (Page 2203)

ComboBox.BorderWidth

Description

The "BorderWidth" property specifies the border width.

Type

UInt8

Access

Read-write

Syntax

```
ComboBox.BorderWidth
```

See also

ComboBox (Page 2203)

ComboBox.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ComboBox.Content
```

See also

ComboBox (Page 2203)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ComboBox.Content (Page 2209)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ComboBox.Content (Page 2209)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ComboBox.Content (Page 2209)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ComboBox.Content (Page 2209)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ComboBox.Content (Page 2209)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ComboBox.Content (Page 2209)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ComboBox.Content (Page 2209)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ComboBox.Content (Page 2209)

ComboBox.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the combo box.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ComboBox.CurrentQuality
```

See also

ComboBox (Page 2203)

ComboBox.Enabled

Description

The "Enabled" property specifies whether the combo box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ComboBox.Enabled
```

See also

ComboBox (Page 2203)

ComboBox.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ComboBox.Font
```

See also

ComboBox (Page 2203)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ComboBox.Font (Page 2216)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ComboBox.Font (Page 2216)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ComboBox.Font (Page 2216)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ComboBox.Font (Page 2216)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ComboBox.Font (Page 2216)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ComboBox.Font (Page 2216)

ComboBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.ForeColor
```

See also

ComboBox (Page 2203)

ComboBox.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.Height
```

See also

ComboBox (Page 2203)

ComboBox.Layer

Description

The "Layer" property returns the screen layer in which the combo box is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ComboBox.Layer
```

See also

ComboBox (Page 2203)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ComboBox.Layer (Page 2220)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ComboBox.Layer (Page 2220)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ComboBox.Layer (Page 2220)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ComboBox.Layer (Page 2220)

ComboBox.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ComboBox.Left
```

See also

ComboBox (Page 2203)

ComboBox.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ComboBox.Margin
```

See also

ComboBox (Page 2203)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ComboBox.Margin (Page 2223)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ComboBox.Margin (Page 2223)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ComboBox.Margin (Page 2223)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ComboBox.Margin (Page 2223)

ComboBox.Name

Description

The "Name" property returns the name of the combo box.

Type

String

Access

Read-only

Syntax

```
ComboBox.Name
```

See also

ComboBox (Page 2203)

ComboBox.Opacity

Description

The "Opacity" property specifies the opacity. The "0" value indicates completely transparency.

Type

Float

Access

Read-write

Syntax

```
ComboBox.Opacity
```

See also

ComboBox (Page 2203)

ComboBox.Operability

Description

The "Operability" property returns whether the combo box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ComboBox.Operability
```

See also

ComboBox (Page 2203)

ComboBox.Padding

Description

The "Padding" property specifies the distance of the content from the border of the combo box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ComboBox.Padding
```

See also

ComboBox (Page 2203)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ComboBox.Padding (Page 2227)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ComboBox.Padding (Page 2227)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ComboBox.Padding (Page 2227)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ComboBox.Padding (Page 2227)

ComboBox.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ComboBox.Parent
```

See also

ComboBox (Page 2203)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ComboBox.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
ComboBox.ProcessValue
```

See also

ComboBox (Page 2203)

ComboBox.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the combo box was created.

Type

String

Access

Read-only

Syntax

```
ComboBox.RenderingTemplate
```

See also

ComboBox (Page 2203)

ComboBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the combo box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ComboBox.RequireExplicitUnlock
```

See also

ComboBox (Page 2203)

ComboBox.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle in degrees.

Type

Int16

Access

Read-write

Syntax

```
ComboBox.RotationAngle
```

See also

ComboBox (Page 2203)

ComboBox.RotationCenterPlacement (Page 2233)

ComboBox.RotationCenterX (Page 2233)

ComboBox.RotationCenterY (Page 2234)

ComboBox.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the combo box rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in the DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
ComboBox.RotationCenterPlacement
```

See also

ComboBox (Page 2203)

ComboBox.RotationAngle (Page 2232)

ComboBox.RotationCenterX (Page 2233)

ComboBox.RotationCenterY (Page 2234)

ComboBox.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
ComboBox.RotationCenterX
```

See also

ComboBox (Page 2203)

ComboBox.RotationAngle (Page 2232)

ComboBox.RotationCenterPlacement (Page 2233)

ComboBox.RotationCenterY (Page 2234)

ComboBox.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
ComboBox.RotationCenterY
```

See also

ComboBox (Page 2203)

ComboBox.RotationAngle (Page 2232)

ComboBox.RotationCenterPlacement (Page 2233)

ComboBox.RotationCenterX (Page 2233)

ComboBox.SelectionItemHeight

Description

The "SelectionItemHeight" property specifies the height of the list entries. The value "0" indicates that the height is calculated automatically.

Type

UInt16

Access

Read-write

Syntax

```
ComboBox.SelectionItemHeight
```

See also

ComboBox (Page 2203)

ComboBox.SelectionItems

Description

The "SelectionItems" property returns the list of all list entries ("SelectionItem" objects) of the combo box.

Type

Object, HmiSelectionItemCollection (Page 2236)

Access

Read-only

Syntax

```
ComboBox.SelectionItems
```

See also

ComboBox (Page 2203)

HmiSelectionItemCollection (Page 2236)

HmiSelectionItemCollection

Description

The "HmiSelectionItemCollection" object is a list of all entries ("SelectionItem" objects) of a list object.

Use

The "HmiSelectionItemCollection" object is a list and can be counted and enumerated. You can access the "HmiSelectionItemCollection" list using the index or the tag name.

Object type

HmiSelectionItemCollection

Properties

The "HmiSelectionItemCollection" object has the following properties:

- **Count**
  Returns the number of list entries of the "HmiSelectionItemCollection" list.

Methods

The "HmiSelectionItemCollection" object has the following methods:

- **Item()**
  Returns a list entry of the "HmiSelectionItemCollection" list.

See also

ComboBox.SelectionItems (Page 2235)

HmiSelectionItemCollection.Count

Description

The "Count" property returns the number of list entries in the "HmiSelectionItemCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiSelectionItemCollection.Count
```

See also

HmiSelectionItemCollection (Page 2236)

HmiSelectionItemCollection.Item()

Description

The "Item" method returns a list entry of the "HmiSelectionItemCollection" list.

Syntax

```
HmiSelectionItemCollection[.Item](HmiSelectionItemName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiSelectionItemCollection" object.

Parameters

`HmiSelectionItemName`

Type: String

Name of the list entry

Return value

Object, HmiSelectionItemPart (Page 2237)

See also

HmiSelectionItemCollection (Page 2236)

SelectionItem (Page 2237)

SelectionItem

Description

The "SelectionItem" object represents a list entry.

Object type

HmiSelectionItemPart

Properties

The "SelectionItem" object has the following properties:

- **Graphic**
  Specifies the graphic of the list entry.
- **IsSelected**
  Specifies whether the list entry is selected.
- **Text**
  Specifies the list entry text.

Methods

--

See also

HmiSelectionItemCollection (Page 2236)

SelectionItem.Graphic

Description

The "Graphic" property specifies the graphic of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Graphic
```

See also

SelectionItem (Page 2237)

SelectionItem.IsSelected

Description

The "IsSelected" property specifies whether the list entry is selected.

Type

Bool

Access

Read-write

Syntax

```
SelectionItem.IsSelected
```

See also

SelectionItem (Page 2237)

SelectionItem.Text

Description

The "Text" property specifies the text of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Text
```

See also

SelectionItem (Page 2237)

ComboBox.SelectionMode

Description

The "SelectionMode" property specifies whether one or more list entries can be selected in the combo box.

Type

Int32, HmiSelectionMode

Specifies the type of the selection:

- NonExclusive (0): Selection of multiple list entries possible
- Exclusive (1): Selection of only one list entry possible

Access

Read-write

Syntax

```
ComboBox.SelectionMode
```

See also

ComboBox (Page 2203)

ComboBox.SelectorPosition

Description

The "SelectorPosition" property specifies the horizontal alignment of the list entries.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
ComboBox.SelectorPosition
```

See also

ComboBox (Page 2203)

ComboBox.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the combo box is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ComboBox.ShowFocusVisual
```

See also

ComboBox (Page 2203)

ComboBox.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the combo box.

Type

String

Access

Read-only

Syntax

```
ComboBox.StyleItemClass
```

See also

ComboBox (Page 2203)

ComboBox.TabIndex

Description

The "TabIndex" property returns the position of the combo box in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ComboBox.TabIndex
```

See also

ComboBox (Page 2203)

ComboBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
ComboBox.ToolTipText
```

See also

ComboBox (Page 2203)

ComboBox.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ComboBox.Top
```

See also

ComboBox (Page 2203)

ComboBox.Visible

Description

The "Visible" property specifies whether the combo box is visible.

Type

Bool

Access

Read-write

Syntax

```
ComboBox.Visible
```

See also

ComboBox (Page 2203)

ComboBox.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
ComboBox.VisualizeQuality
```

See also

ComboBox (Page 2203)

ComboBox.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ComboBox.Width
```

See also

ComboBox (Page 2203)

ComboBox.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the combo box.

Syntax

```
ComboBox.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ComboBox (Page 2203)

ComboBox.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ComboBox.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ComboBox (Page 2203)

ComboBox_OnActivated()

Description

The "OnActivated" event occurs when a combo box receives focus:

- A combo box is selected via the configured tab sequence.
- A combo box that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
ComboBox_OnActivated(item)
```

Context

item

Type: Object

Combo box where the event occurs.

See also

ComboBox (Page 2203)

ComboBox_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A combo box is right-clicked.
- A combo box is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
ComboBox_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Combo box where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ComboBox (Page 2203)

ComboBox_OnDeactivated()

Description

The "OnDeactivated" event occurs when the combo box loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
ComboBox_OnDeactivated(item)
```

Context

item

Type: Object

Combo box where the event occurs.

See also

ComboBox (Page 2203)

ComboBox_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the combo box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
ComboBox_OnKeyDown(item, keyCode, modifiers)
```

Context

`item`

Type: Object

Combo box where the event occurs.

`keyCode`

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

ComboBox (Page 2203)

ComboBox_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the combo box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
ComboBox_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Combo box where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

ComboBox (Page 2203)

ComboBox_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A combo box is left-clicked.
- The <RETURN> or <SPACE> key is pressed when an combo box has the focus.
- A combo box is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
ComboBox_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Combo box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ComboBox (Page 2203)

CommandSourceElement

Description

The "CommandSourceElement" object represents a command source in runtime.

Object type

HmiCommandSourceElement

Properties

The "CommandSourceElement" object has the following properties:

- **Command**
  Returns information about the command.
- **DataConnection**
  Returns the data connection.
- **Parent**
  Returns the higher-level screen object.
- **ResultSet**
  Returns the last result of the executed command.
- **SourceState**
  Returns the state of the data source.

Methods

The "CommandSourceElement" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the command source.
- **PropertyFlashing()**
  Configures flashing of a property.

CommandSourceElement.Command

Description

The "Command" property returns information about the command.

Type

Object, HmiCommandPart

Access

Read-only

Syntax

```
CommandSourceElement.Command
```

See also

CommandSourceElement (Page 2253)

Command.CommandParameters

Description

The "CommandParameters" property returns the parameters of the command.

Type

Object, HmiParameterCollection (Page 2257)

Access

Read-only

Syntax

```
Command.CommandParameters
```

See also

CommandSourceElement.Command (Page 2254)

HmiParameterCollection (Page 2257)

Command.CommandText

Description

The "CommandText" property returns the text of the command.

Type

String

Access

Read-only

Syntax

```
Command.CommandText
```

See also

CommandSourceElement.Command (Page 2254)

Command.CommandType

Description

The "CommandType" property returns the type of the command.

Type

String

Access

Read-only

Syntax

```
Command.CommandType
```

See also

CommandSourceElement.Command (Page 2254)

Command.Execute()

Description

The "Execute" method executes the command.

Syntax

```
Command.Execute()
```

Parameters

--

Return value

--

See also

CommandSourceElement.Command (Page 2254)

HmiParameterCollection

Description

The "HmiParameterCollection" object is a list of all parameters ("Parameter" objects).

Use

The "HmiParameterCollection" object is a list which can be counted and enumerated. You can access the "HmiParameterCollection" list using the index or the tag names.

Object type

HmiParameterCollection

Properties

The "HmiParameterCollection" object has the following properties:

- **Count**
  Returns the number of parameters in the "HmiParameterCollection" list.

Methods

The "HmiParameterCollection" object has the following methods:

- **Item()**
  Returns a parameter of the "HmiParameterCollection" list.

See also

Command.CommandParameters (Page 2255)

HmiParameterCollection.Count

Description

The "Count" property returns the number of parameters in the "HmiParameterCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiParameterCollection.Count
```

See also

HmiParameterCollection (Page 2257)

HmiParameterCollection.Item()

Description

The "Item" method returns a parameter of the "HmiParameterCollection" list.

Syntax

```
HmiParameterCollection.[.Item](HmiParameterName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiParameterCollection" object.

Parameters

`HmiParameterName`

Type: String

Name of the parameter

Return value

Object, HmiParameterPart (Page 2258)

See also

Parameter (Page 2258)

HmiParameterCollection (Page 2257)

Parameter

Description

The "Parameter" object represents a parameter.

Object type

HmiParameterPart

Properties

The "Parameter" object has the following properties:

- **ParameterName**
  Specifies the name of the parameter.
- **ParameterValue**
  Specifies the value of the parameter.

Methods

--

Parameter.ParameterName

Description

The "ParameterName" property specifies the name of the parameter.

Type

String

Access

Read-write

Syntax

```
Parameter.ParameterName
```

See also

Parameter (Page 2258)

Parameter.ParameterValue

Description

The "ParameterValue" property specifies the value of the parameter.

Type

Variant

Access

Read-write

Syntax

```
Parameter.ParameterValue
```

See also

Parameter (Page 2258)

CommandSourceElement.DataConnection

Description

The "DataConnection" property returns the data connection.

Type

Object, HmiDataConnection

Access

Read-only

Syntax

```
CommandSourceElement.DataConnection
```

See also

CommandSourceElement (Page 2253)

CommandSourceElement.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
CommandSourceElement.Parent
```

See also

CommandSourceElement (Page 2253)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

CommandSourceElement.ResultSet

Description

The "ResultSet" property returns the last result of the executed command.

Type

Object, HmiParameterCollection (Page 2262)

Access

Read-only

Syntax

```
CommandSourceElement.ResultSet
```

See also

CommandSourceElement (Page 2253)

HmiParameterCollection (Page 2262)

HmiParameterCollection

Description

The "HmiParameterCollection" object is a list of all parameters ("Parameter" objects).

Use

The "HmiParameterCollection" object is a list which can be counted and enumerated. You can access the "HmiParameterCollection" list using the index or the tag names.

Object type

HmiParameterCollection

Properties

The "HmiParameterCollection" object has the following properties:

- **Count**
  Returns the number of parameters in the "HmiParameterCollection" list.

Methods

The "HmiParameterCollection" object has the following methods:

- **Item()**
  Returns a parameter of the "HmiParameterCollection" list.

See also

CommandSourceElement.ResultSet (Page 2261)

HmiParameterCollection.Count

Description

The "Count" property returns the number of parameters in the "HmiParameterCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiParameterCollection.Count
```

See also

HmiParameterCollection (Page 2262)

HmiParameterCollection.Item()

Description

The "Item" method returns a parameter of the "HmiParameterCollection" list.

Syntax

```
HmiParameterCollection.[.Item](HmiParameterName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiParameterCollection" object.

Parameters

`HmiParameterName`

Type: String

Name of the parameter

Return value

Object, HmiParameterPart (Page 2263)

See also

HmiParameterCollection (Page 2262)

Parameter (Page 2263)

Parameter

Description

The "Parameter" object represents a parameter.

Object type

HmiParameterPart

Properties

The "Parameter" object has the following properties:

- **ParameterName**
  Specifies the name of the parameter.
- **ParameterValue**
  Specifies the value of the parameter.

Methods

--

Parameter.ParameterName

Description

The "ParameterName" property specifies the name of the parameter.

Type

String

Access

Read-write

Syntax

```
Parameter.ParameterName
```

See also

Parameter (Page 2263)

Parameter.ParameterValue

Description

The "ParameterValue" property specifies the value of the parameter.

Type

Variant

Access

Read-write

Syntax

```
Parameter.ParameterValue
```

See also

Parameter (Page 2263)

CommandSourceElement.SourceState

Description

The "SourceState" property returns the state of the data source.

Type

Int32, HmiSourceState

Returns the state:

- Idle (0): Ready
- Busy (1): Busy

Access

Read-only

Syntax

```
CommandSourceElement.SourceState
```

See also

CommandSourceElement (Page 2253)

CommandSourceElement.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the command source.

Syntax

```
CommandSourceElement.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

CommandSourceElement (Page 2253)

CommandSourceElement.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
CommandSourceElement.PropertyFlashing(propertyName, enable[, value]
[, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

CommandSourceElement (Page 2253)

CustomWebControlContainer

Description

The "CustomWebControlContainer" object represents a container as a custom display for user-defined access to web objects in runtime.

The following predefined displays are available:

- AuditAG
- Calendar control
- Performance analyzer
- Performance bar chart
- Performance control
- Performance Gantt chart
- Performance pie chart
- Plant overview
- Reports

Object type

HmiCustomWebControlContainer

Properties

The "CustomWebControlContainer" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the custom display.
- **Enabled**
  Specifies whether the custom display can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the custom display.
- **Layer**
  Returns the screen layer in which the custom display is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.

- **Name**
  Returns the name of the custom display.
- **Operability**
  Returns whether the custom display is operable.
- **Parent**
  Returns the higher-level screen object.
- **Properties**
  Enables access to the dynamic properties of the web objects.
- **RenderingTemplate**
  Returns the name of the template from which the custom display was created.
- **RequireExplicitUnlock**
  Returns whether the custom display is only operable while the corresponding button is being pressed.
- **ShowFocusVisual**
  Specifies whether the custom display is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the custom display.
- **TabIndex**
  Returns the position of the custom display in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the custom display is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the custom display.

Methods

The "CustomWebControlContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the custom display.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "CustomWebControlContainer" object has the following events:

- **OnActivated()**
  Occurs when a custom display receives focus.
- **OnDeactivated()**
  Occurs when a custom display loses focus.

The following events are only available for the "Plant overview" custom display:

- **OnCollapse()**
  Occurs when a custom display is minimized.
- **OnCollapseAll()**
  Occurs when all custom displays are minimized.
- **OnExpand()**
  Occurs when the custom display is expanded.
- **OnExpandAll()**
  Occurs when all custom displays are expanded.
- **OnSelectionChanged()**
  Occurs when the selection is changed.

CustomWebControlContainer.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
CustomWebControlContainer.Authorization
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
CustomWebControlContainer.Caption
```

See also

CustomWebControlContainer (Page 2267)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

CustomWebControlContainer.Caption (Page 2270)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2271)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2271)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2271)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2271)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2271)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2271)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

CustomWebControlContainer.Caption (Page 2270)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

CustomWebControlContainer.Caption (Page 2270)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

CustomWebControlContainer.Caption (Page 2270)

CustomWebControlContainer.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
CustomWebControlContainer.CaptionColor
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects (CustomControl, SwacComponent, or WidgetType).

Type

String

Access

Read-only

Syntax

```
CustomWebControlContainer.ContainedType
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the custom display.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
CustomWebControlContainer.CurrentQuality
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Enabled

Description

The "Enabled" property specifies whether the custom display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
CustomWebControlContainer.Enabled
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
CustomWebControlContainer.Height
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Icon

Description

The "Icon" property specifies the icon of the custom display.

Type

String

Access

Read-write

Syntax

```
CustomWebControlContainer.Icon
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Layer

Description

The "Layer" property returns the screen layer in which the custom display is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
CustomWebControlContainer.Layer
```

See also

CustomWebControlContainer (Page 2267)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

CustomWebControlContainer.Layer (Page 2279)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

CustomWebControlContainer.Layer (Page 2279)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

CustomWebControlContainer.Layer (Page 2279)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

CustomWebControlContainer.Layer (Page 2279)

CustomWebControlContainer.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
CustomWebControlContainer.Left
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
CustomWebControlContainer.Margin
```

See also

CustomWebControlContainer (Page 2267)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

CustomWebControlContainer.Margin (Page 2282)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

CustomWebControlContainer.Margin (Page 2282)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

CustomWebControlContainer.Margin (Page 2282)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

CustomWebControlContainer.Margin (Page 2282)

CustomWebControlContainer.Name

Description

The "Name" property returns the name of the custom display.

Type

String

Access

Read-only

Syntax

```
CustomWebControlContainer.Name
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Operability

Description

The "Operability" property returns whether the custom display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
CustomWebControlContainer.Operability
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
CustomWebControlContainer.Parent
```

See also

CustomWebControlContainer (Page 2267)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

CustomWebControlContainer.Properties

Description

The "Properties" property allows access to the dynamic properties of the web objects.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
CustomWebControlContainer.Properties
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the custom display was created.

Type

String

Access

Read-only

Syntax

```
CustomWebControlContainer.RenderingTemplate
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the custom display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
CustomWebControlContainer.RequireExplicitUnlock
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the custom display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
CustomWebControlContainer.ShowFocusVisual
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the custom display.

Type

String

Access

Read-only

Syntax

```
CustomWebControlContainer.StyleItemClass
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.TabIndex

Description

The "TabIndex" property returns the position of the custom display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
CustomWebControlContainer.TabIndex
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
CustomWebControlContainer.Top
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Visible

Description

The "Visible" property specifies whether the custom display is visible.

Type

Bool

Access

Read-write

Syntax

```
CustomWebControlContainer.Visible
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
CustomWebControlContainer.Width
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the custom display.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
CustomWebControlContainer.WindowFlags
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the custom display.

Syntax

```
CustomWebControlContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
CustomWebControlContainer.PropertyFlashing(propertyName, enable[,
value][, alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnActivated()

Description

The "OnActivated" event occurs when a custom display receives focus:

- A custom display is selected via the configured tab sequence.
- A custom display that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
CustomWebControlContainer_OnActivated(item)
```

Context

item

Type: Object

Custom display where the event occurs.

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnDeactivated()

Description

The "OnDeactivated" event occurs when the custom display loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
CustomWebControlContainer_OnDeactivated(item)
```

Context

item

Type: Object

Custom display where the event occurs.

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnCollapse()

Description

The "OnCollapse" event occurs when a custom display is minimized:

Note

The "OnCollapse" event is only available with the "Plant overview" custom display.

Syntax

```
CustomWebControlContainer_OnCollapse(item, collapsedNode)
```

Context

item

Type: Object

Custom display where the event occurs.

collapsedNode

Type: Object

Custom display which was minimized.

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnCollapseAll()

Description

The "OnCollapseAll" event occurs when all custom displays are minimized.

Note

The "OnCollapseAll" event is only available with the "Plant overview" custom display.

Syntax

```
CustomWebControlContainer_OnCollapseAll(item, isCollapsed)
```

Context

`item`

Type: Object

Custom display where the event occurs.

`isCollapsed`

Type: Object

Returns whether the custom display is minimized.

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnExpand()

Description

The "OnExpand" event occurs when the custom display is expanded.

Note

The "OnExpand" event is only available with the "Plant overview" custom display.

Syntax

```
CustomWebControlContainer_OnExpand(item, expandedNode)
```

Context

item

Type: Object

Custom display where the event occurs.

expandedNode

Type: Object

Custom display which was expanded.

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnExpandAll()

Description

The "OnExpandAll" event occurs when all custom displays are expanded.

Note

The "OnExpandAll" event is only available with the "Plant overview" custom display.

Syntax

```
CustomWebControlContainer_OnExpandAll(item, isExpanded)
```

Context

item

Type: Object

Custom display where the event occurs.

isExpanded

Type: Object

Returns whether the custom display is expanded.

See also

CustomWebControlContainer (Page 2267)

CustomWebControlContainer_OnSelectionChanged()

Description

The "OnSelectionChanged" event occurs when the selection is changed.

Note

The "OnSelectionChanged" event is only available with the "Plant overview" custom display.

Syntax

```
CustomWebControlContainer_OnSelectionChanged(item, selectedNode)
```

Context

`item`

Type: Object

Custom display where the event occurs.

`selectedNode`

Type: Object

Custom display which was selected.

See also

CustomWebControlContainer (Page 2267)

CustomWidgetContainer

Description

The "CustomWidgetContainer" object represents a container as custom display in runtime.

Object type

HmiCustomWidgetContainer

Properties

The "CustomWidgetContainer" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the custom display.
- **Enabled**
  Specifies whether the custom display can be operated in runtime.
- **Height**
  Specifies the height.
- **Layer**
  Returns the screen layer in which the custom display is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the custom display.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the custom display is operable.
- **Parent**
  Returns the higher-level screen object.
- **Properties**
  Enables access to the dynamic properties of the web objects in the customized display.
- **RenderingTemplate**
  Returns the name of the template from which the custom display was created.
- **RequireExplicitUnlock**
  Returns whether the custom display is only operable while the corresponding button is being pressed.
- **RotationAngle**
  Specifies the rotation angle in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the custom display rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.

- **ShowFocusVisual**
  Specifies whether the custom display is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the custom display.
- **TabIndex**
  Returns the position of the custom display in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the custom display is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "CustomWidgetContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the custom display.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "CustomWidgetContainer" object has the following events:

- **OnActivated()**
  Occurs when a custom display receives focus.
- **OnContextTapped()**
  Occurs when a custom display is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a custom display loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the custom display is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the custom display is in focus.
- **OnTapped()**
  Occurs when a custom display is left-clicked or short-touched.

CustomWidgetContainer.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
CustomWidgetContainer.Authorization
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects (CustomControl, SwacComponent, or WidgetType).

Type

String

Access

Read-only

Syntax

```
CustomWidgetContainer.ContainedType
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the custom display.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
CustomWidgetContainer.CurrentQuality
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Enabled

Description

The "Enabled" property specifies whether the custom display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
CustomWidgetContainer.Enabled
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
CustomWidgetContainer.Height
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Layer

Description

The "Layer" property returns the screen layer in which the custom display is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
CustomWidgetContainer.Layer
```

See also

CustomWidgetContainer (Page 2298)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

CustomWidgetContainer.Layer (Page 2303)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

CustomWidgetContainer.Layer (Page 2303)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

CustomWidgetContainer.Layer (Page 2303)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

CustomWidgetContainer.Layer (Page 2303)

CustomWidgetContainer.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
CustomWidgetContainer.Left
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
CustomWidgetContainer.Margin
```

See also

CustomWidgetContainer (Page 2298)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

CustomWidgetContainer.Margin (Page 2306)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

CustomWidgetContainer.Margin (Page 2306)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

CustomWidgetContainer.Margin (Page 2306)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

CustomWidgetContainer.Margin (Page 2306)

CustomWidgetContainer.Name

Description

The "Name" property returns the name of the custom display.

Type

String

Access

Read-only

Syntax

```
CustomWidgetContainer.Name
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Opacity

Description

The "Opacity" property specifies the opacity. The "0" value indicates completely transparency.

Type

Float

Access

Read-write

Syntax

```
CustomWidgetContainer.Opacity
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Operability

Description

The "Operability" property returns whether the custom display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
CustomWidgetContainer.Operability
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
CustomWidgetContainer.Parent
```

See also

CustomWidgetContainer (Page 2298)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

CustomWidgetContainer.Properties

Description

The "Properties" property allows access to the dynamic properties of the objects in the customized display.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
CustomWidgetContainer.Properties
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the custom display was created.

Type

String

Access

Read-only

Syntax

```
CustomWidgetContainer.RenderingTemplate
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the custom display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
CustomWidgetContainer.RequireExplicitUnlock
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle in degrees.

Type

Int16

Access

Read-write

Syntax

```
CustomWidgetContainer.RotationAngle
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RotationCenterPlacement (Page 2313)

CustomWidgetContainer.RotationCenterX (Page 2314)

CustomWidgetContainer.RotationCenterY (Page 2314)

CustomWidgetContainer.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the custom display rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in the DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
CustomWidgetContainer.RotationCenterPlacement
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RotationAngle (Page 2312)

CustomWidgetContainer.RotationCenterX (Page 2314)

CustomWidgetContainer.RotationCenterY (Page 2314)

CustomWidgetContainer.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CustomWidgetContainer.RotationCenterX
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RotationAngle (Page 2312)

CustomWidgetContainer.RotationCenterPlacement (Page 2313)

CustomWidgetContainer.RotationCenterY (Page 2314)

CustomWidgetContainer.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
CustomWidgetContainer.RotationCenterY
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.RotationAngle (Page 2312)

CustomWidgetContainer.RotationCenterPlacement (Page 2313)

CustomWidgetContainer.RotationCenterX (Page 2314)

CustomWidgetContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the custom display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
CustomWidgetContainer.ShowFocusVisual
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the custom display.

Type

String

Access

Read-only

Syntax

```
CustomWidgetContainer.StyleItemClass
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.TabIndex

Description

The "TabIndex" property returns the position of the custom display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
CustomWidgetContainer.TabIndex
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
CustomWidgetContainer.ToolTipText
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
CustomWidgetContainer.Top
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Visible

Description

The "Visible" property specifies whether the custom display is visible.

Type

Bool

Access

Read-write

Syntax

```
CustomWidgetContainer.Visible
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
CustomWidgetContainer.VisualizeQuality
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
CustomWidgetContainer.Width
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the custom display.

Syntax

```
CustomWidgetContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
CustomWidgetContainer.PropertyFlashing(propertyName, enable[, value]
[, alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer_OnActivated()

Description

The "OnActivated" event occurs when a custom display receives focus:

- A custom display is selected via the configured tab sequence.
- A custom display that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
CustomWidgetContainer_OnActivated(item)
```

Context

item

Type: Object

Custom display where the event occurs.

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A custom display is right-clicked.
- A custom display is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
CustomWidgetContainer_OnContextTapped(item, x, y, modifiers,
trigger)
```

Context

`item`

Type: Object

Custom display where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer_OnDeactivated()

Description

The "OnDeactivated" event occurs when the custom display loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
CustomWidgetContainer_OnDeactivated(item)
```

Context

item

Type: Object

Custom display where the event occurs.

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the custom display is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CustomWidgetContainer_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Custom display where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the custom display is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
CustomWidgetContainer_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Custom display where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

CustomWidgetContainer (Page 2298)

CustomWidgetContainer_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A custom display is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a custom display has the focus.
- A custom display is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
CustomWidgetContainer_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Custom display where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

CustomWidgetContainer (Page 2298)

DataGridControl

Description

The "DataGridControl" object represents a data table display for showing tag values in runtime.

Object type

HmiDataGridControl

Properties

The "DataGridControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **BindingSource**
  Returns the data source.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the data table display.
- **DataGridView**
  Specifies the appearance of the data table.
- **Enabled**
  Specifies whether the data table display can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the data table display.
- **Layer**
  Returns the screen layer in which the data table display is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the data table display.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the data table display was created.
- **ShowFocusVisual**
  Specifies whether the data table display is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the data table display.
- **StyleItemClass**
  Returns the style which is applied to the data table display.
- **TabIndex**
  Returns the position of the data table display in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the data table display.

- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the data table display is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the data table display.

Methods

The "DataGridControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the data table display.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "DataGridControl" object has the following events:

- **OnActivated()**
  Occurs when a data table display receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the data table display.
- **OnDeactivated()**
  Occurs when a data table display loses focus.
- **OnInitialized()**
  Occurs when a data table display has been successfully initialized and the data connection to the PLC has been established.

DataGridControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridControl.BackColor
```

See also

DataGridControl (Page 2327)

DataGridControl.BindingSource

Description

The "BindingSource" property returns the data source.

Type

Object, HmiBindingSourceElement (Page 1943)

Access

Read-only

Syntax

```
DataGridControl.BindingSource
```

See also

DataGridControl (Page 2327)

BindingSourceElement (Page 1943)

BindingSourceElement

Description

BindingSourceElement (Page 1943)

DataGridControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
DataGridControl.Caption
```

See also

DataGridControl (Page 2327)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

DataGridControl.Caption (Page 2330)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2331)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2331)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2331)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2331)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2331)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2331)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

DataGridControl.Caption (Page 2330)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

DataGridControl.Caption (Page 2330)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

DataGridControl.Caption (Page 2330)

DataGridControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
DataGridControl.CaptionColor
```

See also

DataGridControl (Page 2327)

DataGridControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the data table display.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
DataGridControl.CurrentQuality
```

See also

DataGridControl (Page 2327)

DataGridControl.DataGridView

Description

The "DataGridView" property specifies the appearance of the data table.

Type

Object, HmiDataGridViewPart (Page 2338)

Access

Read-write

Syntax

```
DataGridControl.DataGridView
```

See also

DataGridControl (Page 2327)

DataGridView (Page 2338)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

DataGridControl.DataGridView (Page 2337)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 2341)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 2341)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 2341)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 2341)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 2344)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

DataGridControl.DataGridView (Page 2337)

HmiDataGridColumnCollection (Page 2344)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 2344)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 2346)

See also

HmiDataGridColumnCollection (Page 2344)

DataGridColumn (Page 2346)

DataGridColumn

Description

The "DataGridColumn" object represents a value column.

Object type

HmiDataGridColumnPart

Properties

The "DataGridColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies display options for text and graphics.

- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **Key**
  Corresponds to the column definition from the "ConsideredColumns" property of the connected source.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width of the column in the DIU (Device Independent Unit).

Methods

--

See also

HmiDataGridColumnCollection (Page 2344)

DataGridColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
DataGridColumn.AllowSort
```

See also

DataGridColumn (Page 2346)

DataGridColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridColumn.BackColor
```

See also

DataGridColumn (Page 2346)

DataGridColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumn.Content
```

See also

DataGridColumn (Page 2346)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumn.Content (Page 2348)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumn.Content (Page 2348)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered

- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumn.Content (Page 2348)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumn.Content (Page 2348)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumn.Content (Page 2348)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumn.Content (Page 2348)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumn.Content (Page 2348)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumn.Content (Page 2348)

DataGridColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
DataGridColumn.Enabled
```

See also

DataGridColumn (Page 2346)

DataGridColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridColumn.ForeColor
```

See also

DataGridColumn (Page 2346)

DataGridColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
DataGridColumn.Header
```

See also

DataGridColumn (Page 2346)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

DataGridColumn.Header (Page 2355)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 2355)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 2355)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

DataGridColumn.Header (Page 2355)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

DataGridColumn.Header (Page 2355)

DataGridColumn.Key

Description

The "Key" property corresponds to the column definition from the "ConsideredColumns" property of the connected source.

Type

String

Access

Read-only

Syntax

```
DataGridColumn.Key
```

See also

DataGridColumn (Page 2346)

DataGridColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
DataGridColumn.MaximumWidth
```

See also

DataGridColumn (Page 2346)

DataGridColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
DataGridColumn.MinimumWidth
```

See also

DataGridColumn (Page 2346)

DataGridColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
DataGridColumn.Name
```

See also

DataGridColumn (Page 2346)

DataGridColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
DataGridColumn.OutputFormat
```

See also

DataGridColumn (Page 2346)

DataGridColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
DataGridColumn.SortDirection
```

See also

DataGridColumn (Page 2346)

DataGridColumn.SortOrder

Description

The "SortOrder" property specifies the sorting order.

The sorting index begins at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
DataGridColumn.SortOrder
```

See also

DataGridColumn (Page 2346)

DataGridColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
DataGridColumn.Visible
```

See also

DataGridColumn (Page 2346)

DataGridColumn.Width

Description

The "Width" property specifies the width of the column in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DataGridColumn.Width
```

See also

DataGridColumn (Page 2346)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

DataGridControl.DataGridView (Page 2337)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 2366)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 2366)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 2366)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 2366)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 2366)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 2366)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 2373)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 2375)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 2375)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 2375)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 2375)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 2375)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 2375)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

Object

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 2373)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 2373)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.SelectionBorderWidth

Description

The "SelectionBorderWidth" property specifies the border width of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderWidth
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

DataGridControl.DataGridView (Page 2337)

DataGridControl.Enabled

Description

The "Enabled" property specifies whether the data table display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
DataGridControl.Enabled
```

See also

DataGridControl (Page 2327)

DataGridControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DataGridControl.Height
```

See also

DataGridControl (Page 2327)

DataGridControl.Icon

Description

The "Icon" property specifies the icon of the data table display.

Type

String

Access

Read-write

Syntax

```
DataGridControl.Icon
```

See also

DataGridControl (Page 2327)

DataGridControl.Layer

Description

The "Layer" property returns the screen layer in which the data tables display is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
DataGridControl.Layer
```

See also

DataGridControl (Page 2327)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

DataGridControl.Layer (Page 2388)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

DataGridControl.Layer (Page 2388)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

DataGridControl.Layer (Page 2388)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

DataGridControl.Layer (Page 2388)

DataGridControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DataGridControl.Left
```

See also

DataGridControl (Page 2327)

DataGridControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
DataGridControl.Margin
```

See also

DataGridControl (Page 2327)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

DataGridControl.Margin (Page 2391)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

DataGridControl.Margin (Page 2391)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

DataGridControl.Margin (Page 2391)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

DataGridControl.Margin (Page 2391)

DataGridControl.Name

Description

The "Name" property returns the name of the data table display.

Type

String

Access

Read-only

Syntax

```
DataGridControl.Name
```

See also

DataGridControl (Page 2327)

DataGridControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
DataGridControl.Parent
```

See also

DataGridControl (Page 2327)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

DataGridControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the data table display was created.

Type

String

Access

Read-only

Syntax

```
DataGridControl.RenderingTemplate
```

See also

DataGridControl (Page 2327)

DataGridControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the data table display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
DataGridControl.ShowFocusVisual
```

See also

DataGridControl (Page 2327)

DataGridControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the data table display.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
DataGridControl.StatusBar
```

See also

DataGridControl (Page 2327)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

DataGridControl.StatusBar (Page 2395)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 2397)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

DataGridControl.StatusBar (Page 2395)

HmiControlBarElementCollection (Page 2397)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 2396)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 2397)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 2414)

See also

HmiControlBarElementCollection (Page 2397)

Control Bar Elements (Page 2414)

Control Bar Elements

Description

Control Bar Elements (Page 2414)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

DataGridControl.StatusBar (Page 2395)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

DataGridControl.StatusBar (Page 2395)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 2399)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 2399)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 2399)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 2399)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 2399)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 2399)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

DataGridControl.StatusBar (Page 2395)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 2403)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 2403)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 2403)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 2403)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

DataGridControl.StatusBar (Page 2395)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 2406)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 2406)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 2406)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 2406)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

DataGridControl.StatusBar (Page 2395)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

DataGridControl.StatusBar (Page 2395)

DataGridControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the data table display.

Type

String

Access

Read-only

Syntax

```
DataGridControl.StyleItemClass
```

See also

DataGridControl (Page 2327)

DataGridControl.TabIndex

Description

The "TabIndex" property returns the position of the data table display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
DataGridControl.TabIndex
```

See also

DataGridControl (Page 2327)

DataGridControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the data table display.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
DataGridControl.ToolBar
```

See also

DataGridControl (Page 2327)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

DataGridControl.ToolBar (Page 2410)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 2412)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

DataGridControl.ToolBar (Page 2410)

HmiControlBarElementCollection (Page 2412)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 2411)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 2412)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 2414)

See also

HmiControlBarElementCollection (Page 2412)

Control Bar Elements (Page 2414)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.

- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 2414)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 2414)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 2414)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 2414)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 2414)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 2419)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 2419)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 2419)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 2419)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 2419)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 2419)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 2419)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 2419)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 2414)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 2414)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 2414)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 2430)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 2430)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 2430)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 2430)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 2414)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 2414)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 2414)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 2414)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 2435)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 2435)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 2435)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 2435)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 2414)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 2414)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 2414)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.

- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 2440)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 2442)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 2442)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 2442)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 2442)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 2442)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 2442)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 2442)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 2442)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 2440)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 2452)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 2452)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 2452)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 2452)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 2440)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 2457)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 2457)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 2457)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 2457)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 2440)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 2440)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 2462)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 2469)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 2469)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 2469)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 2469)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 2462)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 2474)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 2474)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 2474)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 2474)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 2462)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 2462)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 2479)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 2485)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 2485)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 2485)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 2485)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 2479)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 2490)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 2490)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 2490)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 2490)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 2479)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 2479)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 2495)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 2504)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 2504)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 2504)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 2504)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 2495)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 2509)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 2509)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 2509)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 2509)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 2495)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 2495)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 2515)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 2522)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 2522)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 2515)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 2533)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 2533)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 2533)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 2533)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 2515)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 2538)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 2538)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 2538)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 2538)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 2515)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 2515)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

DataGridControl.ToolBar (Page 2410)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

DataGridControl.ToolBar (Page 2410)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 2544)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 2544)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 2544)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 2544)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 2544)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 2544)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

DataGridControl.ToolBar (Page 2410)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 2547)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 2547)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 2547)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 2547)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

DataGridControl.ToolBar (Page 2410)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 2550)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 2550)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 2550)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 2550)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

DataGridControl.ToolBar (Page 2410)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

DataGridControl.ToolBar (Page 2410)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

DataGridControl.ToolBar (Page 2410)

DataGridControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DataGridControl.Top
```

See also

DataGridControl (Page 2327)

DataGridControl.Visible

Description

The "Visible" property specifies whether the data table display is visible.

Type

Bool

Access

Read-write

Syntax

```
DataGridControl.Visible
```

See also

DataGridControl (Page 2327)

DataGridControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DataGridControl.Width
```

See also

DataGridControl (Page 2327)

DataGridControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the data table display.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title

- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
DataGridControl.WindowFlags
```

Example

Adapting the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

DataGridControl (Page 2327)

DataGridControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the data table display.

Syntax

```
DataGridControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

DataGridControl (Page 2327)

DataGridControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the data table display.

Syntax

```
DataGridControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

DataGridControl (Page 2327)

DataGridControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
DataGridControl.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

DataGridControl (Page 2327)

DataGridControl_OnActivated()

Description

The "OnActivated" event occurs when a data table display receives focus:

- A data table display is selected via the configured tab sequence.
- A data table display that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
DataGridControl_OnActivated(item)
```

Context

item

Type: Object

Data table display where the event occurs.

See also

DataGridControl (Page 2327)

DataGridControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the data table display.

Syntax

```
DataGridControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Data table display where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

DataGridControl (Page 2327)

DataGridControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when the data table display loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
DataGridControl_OnDeactivated(item)
```

Context

item

Type: Object

Data table display where the event occurs.

See also

DataGridControl (Page 2327)

DataGridControl_OnInitialized()

Description

The "OnInitialized" event occurs when a data table display has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
DataGridControl_OnInitialized(item)
```

Context

item

Type: Object

Data table display where the event occurs.

See also

DataGridControl (Page 2327)

DcsFaceplateContainer

Description

The "DcsFaceplateContainer" object represents a container for faceplate instances that can only be used by DCS+ products.

Object type

HmiDcsFaceplateContainer

Properties

The "DcsFaceplateContainer" object has the following properties:

- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the faceplate container.
- **DomainSubTypeGUID**
  Returns the unique designation for the subtype of the displayed faceplate container.
- **DomainTypeGUID**
  Returns the unique designation for the type of displayed faceplate container.
- **Enabled**
  Specifies whether the faceplate container can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon.
- **Layer**
  Returns the layer of the screen in which the faceplate container is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the faceplate container.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the faceplate container was created.
- **ShowFocusVisual**
  Specifies whether the faceplate container is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the faceplate container.
- **TabIndex**
  Returns the position of the faceplate container in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the faceplate container is visible.

- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the faceplate container.

Methods

The "DcsFaceplateContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the faceplate container.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "DcsFaceplateContainer" object has the following events:

- **OnActivated()**
  Occurs when a faceplate container receives focus.
- **OnDeactivated()**
  Occurs when a faceplate container loses focus.

DcsFaceplateContainer.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
DcsFaceplateContainer.Caption
```

See also

DcsFaceplateContainer (Page 2561)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

DcsFaceplateContainer.Caption (Page 2563)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2564)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2564)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2564)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2564)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2564)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2564)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

DcsFaceplateContainer.Caption (Page 2563)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

DcsFaceplateContainer.Caption (Page 2563)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

DcsFaceplateContainer.Caption (Page 2563)

DcsFaceplateContainer.CaptionColor

Description

The "CaptionColor" property specifies the background color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
DcsFaceplateContainer.CaptionColor
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects.

Type

String

Access

Read-only

Syntax

```
DcsFaceplateContainer.ContainedType
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the faceplate container.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
DcsFaceplateContainer.CurrentQuality
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.DomainSubTypeGUID

Description

The "DomainSubTypeGUID" property returns the unique designation for the subtype of the displayed faceplate container.

Type

String

Access

Read-only

Syntax

```
DcsFaceplateContainer.DomainSubTypeGUID
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.DomainTypeGUID

Description

The "DomainTypeGUID" property returns the unique designation for the type of the displayed faceplate container.

Type

String

Access

Read-only

Syntax

```
DcsFaceplateContainer.DomainTypeGUID
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Enabled

Description

The "Enabled" property specifies whether the faceplate container can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
DcsFaceplateContainer.Enabled
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DcsFaceplateContainer.Height
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Icon

Description

The "Icon" property specifies the icon.

Type

String

Access

Read-write

Syntax

```
DcsFaceplateContainer.Icon
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Layer

Description

The "Layer" property returns the layer of the screen in which the faceplate container is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
DcsFaceplateContainer.Layer
```

See also

DcsFaceplateContainer (Page 2561)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

DcsFaceplateContainer.Layer (Page 2573)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

DcsFaceplateContainer.Layer (Page 2573)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

DcsFaceplateContainer.Layer (Page 2573)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

DcsFaceplateContainer.Layer (Page 2573)

DcsFaceplateContainer.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DcsFaceplateContainer.Left
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Margin

Description

The Margin property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
DcsFaceplateContainer.Margin
```

See also

DcsFaceplateContainer (Page 2561)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

DcsFaceplateContainer.Margin (Page 2576)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

DcsFaceplateContainer.Margin (Page 2576)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

DcsFaceplateContainer.Margin (Page 2576)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

DcsFaceplateContainer.Margin (Page 2576)

DcsFaceplateContainer.Name

Description

The "Name" property returns the name of the faceplate container.

Type

String

Access

Read-only

Syntax

```
DcsFaceplateContainer.Name
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
DcsFaceplateContainer.Parent
```

See also

DcsFaceplateContainer (Page 2561)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

DcsFaceplateContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the faceplate container was created.

Type

String

Access

Read-only

Syntax

```
DcsFaceplateContainer.RenderingTemplate
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the faceplate container is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
DcsFaceplateContainer.ShowFocusVisual
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the faceplate container.

Type

String

Access

Read-only

Syntax

```
DcsFaceplateContainer.StyleItemClass
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.TabIndex

Description

The "TabIndex" property returns the position of the faceplate container in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
DcsFaceplateContainer.TabIndex
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DcsFaceplateContainer.Top
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Visible

Description

The "Visible" property specifies whether the faceplate container is visible.

Type

Bool

Access

Read-write

Syntax

```
DcsFaceplateContainer.Visible
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DcsFaceplateContainer.Width
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the faceplate container.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
DcsFaceplateContainer.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the faceplate container.

Syntax

```
DcsFaceplateContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
DcsFaceplateContainer.PropertyFlashing(propertyName, enable[, value]
[, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer_OnActivated()

Description

The "OnActivated" event occurs when a faceplate container receives focus:

- A faceplate container is selected via the configured tab sequence.
- A faceplate container that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
DcsFaceplateContainer_OnActivated(item)
```

Context

item

Type: Object

Faceplate container where the event occurs.

See also

DcsFaceplateContainer (Page 2561)

DcsFaceplateContainer_OnDeactivated()

Description

The "OnDeactivated" event occurs when a faceplate container loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
DcsFaceplateContainer_OnDeactivated(item)
```

Context

item

Type: Object

Faceplate container where the event occurs.

See also

DcsFaceplateContainer (Page 2561)

DetailedParameterControl

Description

The "DetailedParameterControl" object represents a parameter set display of tag values from the current process or the archive in runtime.

Object type

HmiDetailedParameterControl

Properties

The "DetailedParameterControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentParameterSetID**
  Specifies the current parameter set ID.
- **CurrentParameterSetTypeID**
  Specifies the current parameter set type ID.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the parameter set display.
- **EditMode**
  Specifies the editing mode for values in runtime.
- **Enabled**
  Specifies whether the parameter set display can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **HideDetails**
  Specifies whether the table for value display is hidden.
- **Icon**
  Specifies the icon of the parameter set display.
- **Layer**
  Returns the screen layer in which the parameter set display is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the parameter set display.
- **ParameterSetTypeFixed**
  Specifies a parameter set type which cannot be changed in runtime.
- **ParameterView**
  Defines the appearance of the parameter table.
- **Parent**
  Returns the higher-level screen object.

- **RenderingTemplate**
  Returns the name of the template from which the parameter set display was created.
- **SelectionBackColor**
  Specifies the background color of the selected cells.
- **SelectionForeColor**
  Specifies the foreground color of the selected cells.
- **ShowFocusVisual**
  Specifies whether the parameter set display is highlighted when in focus.
- **StatusBar**
  Sets the information bar of the parameter set display.
- **StyleItemClass**
  Returns the style which is applied to the parameter set display.
- **TabIndex**
  Returns the position of the parameter set display in the tab sequence.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Sets the toolbar of the parameter set display.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the parameter set display is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the parameter set display.

Methods

The "DetailedParameterControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the parameter set display.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "DetailedParameterControl" object has the following events:

- **OnActivated()**
  Occurs when a parameter set display receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the parameter set display.
- **OnDeactivated()**
  Occurs when a parameter set display loses focus.
- **OnInitialized()**
  Occurs when a parameter set display has been successfully initialized and the data connection to the PLC has been established.

DetailedParameterControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.BackColor
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
DetailedParameterControl.Caption
```

See also

DetailedParameterControl (Page 2587)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

DetailedParameterControl.Caption (Page 2590)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2591)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2591)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2591)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2591)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2591)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2591)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

DetailedParameterControl.Caption (Page 2590)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

DetailedParameterControl.Caption (Page 2590)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

DetailedParameterControl.Caption (Page 2590)

DetailedParameterControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.CaptionColor
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.CurrentParameterSetID

Description

The "CurrentParameterSetID" property specifies the current parameter set ID.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.CurrentParameterSetID
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.CurrentParameterSetTypeID

Description

The "CurrentParameterSetTypeID" property specifies the current parameter set ID.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.CurrentParameterSetTypeID
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the parameter set display.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
DetailedParameterControl.CurrentQuality
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.EditMode

Description

The "EditMode" property specifies the editing mode for values in runtime.

Type

Int32, HmiEditMode

Specifies the editing mode:

- None (0): No access
- Update (1): Update values

- Create (2): Create values
- Delete (4): Delete values

Access

Read-write

Syntax

```
DetailedParameterControl.EditMode
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Enabled

Description

The "Enabled" property specifies whether the parameter set display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControl.Enabled
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DetailedParameterControl.Font
```

See also

DetailedParameterControl (Page 2587)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DetailedParameterControl.Font (Page 2599)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DetailedParameterControl.Font (Page 2599)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DetailedParameterControl.Font (Page 2599)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DetailedParameterControl.Font (Page 2599)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DetailedParameterControl.Font (Page 2599)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DetailedParameterControl.Font (Page 2599)

DetailedParameterControl.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.ForeColor
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.Height
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.HideDetails

Description

The "HideDetails" property specifies whether the table is hidden for the value display.

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControl.HideDetails
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Icon

Description

The "Icon" property specifies the icon of the parameter set display.

Type

String

Access

Read-write

Syntax

```
DetailedParameterControl.Icon
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Layer

Description

The "Layer" property returns the layer of the screen in which the parameter set display is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
DetailedParameterControl.Layer
```

See also

DetailedParameterControl (Page 2587)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

DetailedParameterControl.Layer (Page 2605)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

DetailedParameterControl.Layer (Page 2605)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

DetailedParameterControl.Layer (Page 2605)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

DetailedParameterControl.Layer (Page 2605)

DetailedParameterControl.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DetailedParameterControl.Left
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
DetailedParameterControl.Margin
```

See also

DetailedParameterControl (Page 2587)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

DetailedParameterControl.Margin (Page 2608)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

DetailedParameterControl.Margin (Page 2608)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

DetailedParameterControl.Margin (Page 2608)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

DetailedParameterControl.Margin (Page 2608)

DetailedParameterControl.Name

Description

The "Name" property returns the parameter set display of the column.

Type

String

Access

Read-only

Syntax

```
DetailedParameterControl.Name
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.ParameterSetTypeFixed

Description

The "ParameterSetTypeFixed" property specifies a parameter set type which cannot be changed in runtime.

Type

String, HmiParameterSetType (Page 1234)

Access

Read-write

Syntax

```
DetailedParameterControl.ParameterSetTypeFixed
```

See also

DetailedParameterControl (Page 2587)

ParameterSetType (Page 1234)

ParameterSetType

Description

ParameterSetType (Page 1234)

DetailedParameterControl.ParameterView

Description

The "ParameterView" property defines the appearance of the parameter table.

Type

Object, HmiDataGridViewPart (Page 2612)

Access

Read-write

Syntax

```
DetailedParameterControl.ParameterView
```

See also

DetailedParameterControl (Page 2587)

DataGridView (Page 2612)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

DetailedParameterControl.ParameterView (Page 2612)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 2615)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 2615)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 2615)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 2615)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 2619)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

DetailedParameterControl.ParameterView (Page 2612)
HmiDataGridColumnCollection (Page 2619)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

See also

DataGridView.Columns (Page 2618)

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 2619)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 2621)

See also

HmiDataGridColumnCollection (Page 2619)

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn

Description

The "DetailedParameterControlColumn" object represents a value column.

Object type

HmiDetailedParameterControlColumnPart

Properties

The "DetailedParameterControlColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies the display options for text and graphics.
- **DetailedParameterControlBlock**
  Specifies the information blocks.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **Key**
  Corresponds to the column definition from the "ConsideredColumns" property of the connected source.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.

- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection (Page 2619)

DetailedParameterControlColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControlColumn.AllowSort
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControlColumn.BackColor
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DetailedParameterControlColumn.Content
```

See also

DetailedParameterControlColumn (Page 2621)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics

- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DetailedParameterControlColumn.Content (Page 2623)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DetailedParameterControlColumn.Content (Page 2623)

DetailedParameterControlColumn.DetailedParameterControlBlock

Description

The "DetailedParameterControlBlock" property specifies the information blocks.

Type

Int32, HmiDetailedParameterControlBlock

Specifies the information blocks:

- None (0): None
- ParameterSetElementName (1): Unit name, for example "kilogram", must always be set
- ParameterSetValue (2): Value, must always be set
- ParameterSetElementUnit (3): Unit, for example "kg"

Access

Read-write

Syntax

```
DetailedParameterControlColumn.DetailedParameterControlBlock
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControlColumn.Enabled
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControlColumn.ForeColor
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
DetailedParameterControlColumn.Header
```

See also

DetailedParameterControlColumn (Page 2621)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

DetailedParameterControlColumn.Header (Page 2630)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 2630)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 2630)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

DetailedParameterControlColumn.Header (Page 2630)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

DetailedParameterControlColumn.Header (Page 2630)

DetailedParameterControlColumn.Key

Description

The "Key" property corresponds to the column definition from the "ConsideredColumns" property of the connected source.

Type

String

Access

Read-only

Syntax

```
DetailedParameterControlColumn.Key
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControlColumn.MaximumWidth
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControlColumn.MinimumWidth
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
DetailedParameterControlColumn.Name
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
DetailedParameterControlColumn.OutputFormat
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
DetailedParameterControlColumn.SortDirection
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.SortOrder

Description

The "SortOrder" property specifies the sorting order.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
DetailedParameterControlColumn.SortOrder
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControlColumn.Visible
```

See also

DetailedParameterControlColumn (Page 2621)

DetailedParameterControlColumn.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControlColumn.Width
```

See also

DetailedParameterControlColumn (Page 2621)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

DetailedParameterControl.ParameterView (Page 2612)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 2641)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 2641)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 2641)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 2641)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 2641)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 2641)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 2647)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 2649)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 2649)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 2649)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 2649)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 2649)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 2649)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 2647)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 2647)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

DetailedParameterControl.ParameterView (Page 2612)

DetailedParameterControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
DetailedParameterControl.Parent
```

See also

DetailedParameterControl (Page 2587)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

DetailedParameterControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the parameter set display was created.

Type

String

Access

Read-only

Syntax

```
DetailedParameterControl.RenderingTemplate
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.SelectionBackColor
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.SelectionForeColor

Description

The "SelectionForeColor" property specifies the foreground color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.SelectionForeColor
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the parameter set display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControl.ShowFocusVisual
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the parameter set display.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
DetailedParameterControl.StatusBar
```

See also

DetailedParameterControl (Page 2587)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

DetailedParameterControl.StatusBar (Page 2663)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 2664)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

DetailedParameterControl.StatusBar (Page 2663)

HmiControlBarElementCollection (Page 2664)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 2664)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 2664)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 2682)

See also

HmiControlBarElementCollection (Page 2664)

Control Bar Elements (Page 2682)

Control Bar Elements

Description

Control Bar Elements (Page 2682)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

DetailedParameterControl.StatusBar (Page 2663)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

DetailedParameterControl.StatusBar (Page 2663)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 2667)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 2667)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 2667)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 2667)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 2667)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 2667)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

DetailedParameterControl.StatusBar (Page 2663)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 2671)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 2671)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 2671)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 2671)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

DetailedParameterControl.StatusBar (Page 2663)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 2673)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 2673)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 2673)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 2673)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

DetailedParameterControl.StatusBar (Page 2663)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

DetailedParameterControl.StatusBar (Page 2663)

DetailedParameterControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the parameter set display.

Type

String

Access

Read-only

Syntax

```
DetailedParameterControl.StyleItemClass
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.TabIndex

Description

The "TabIndex" property returns the position of the parameter set display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
DetailedParameterControl.TabIndex
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.TimeZone

Description

The "TimeZone" property specifies the time zone.

Type

Int32, HmiTimeZone

Access

Read-write

Syntax

```
DetailedParameterControl.TimeZone
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the parameter set display.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
DetailedParameterControl.ToolBar
```

See also

DetailedParameterControl (Page 2587)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

DetailedParameterControl.ToolBar (Page 2678)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 2680)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

DetailedParameterControl.ToolBar (Page 2678)

HmiControlBarElementCollection (Page 2680)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 2679)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 2680)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 2682)

See also

HmiControlBarElementCollection (Page 2680)

Control Bar Elements (Page 2682)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.

- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 2682)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 2682)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 2682)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 2682)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 2682)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 2687)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 2687)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 2687)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 2687)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 2687)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above

- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 2687)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 2687)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 2687)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 2682)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 2682)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog

- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled

- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 2682)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 2697)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 2697)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 2697)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 2697)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 2682)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 2682)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 2682)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 2682)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 2703)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 2703)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 2703)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 2703)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 2682)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 2682)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 2682)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 2708)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 2710)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 2710)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 2710)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 2710)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 2710)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 2710)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 2710)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 2710)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 2708)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 2720)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 2720)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 2720)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 2720)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 2708)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 2725)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 2725)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 2725)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 2725)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 2708)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 2708)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 2730)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 2737)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 2737)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 2737)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 2737)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 2730)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 2742)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 2742)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 2742)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 2742)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 2730)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 2730)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 2747)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 2753)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 2753)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 2753)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 2753)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 2747)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 2758)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 2758)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 2758)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 2758)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 2747)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 2747)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 2763)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 2772)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 2772)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 2772)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 2772)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 2763)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 2777)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 2777)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 2777)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 2777)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 2763)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 2763)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 2783)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 2790)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 2790)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 2783)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 2801)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 2801)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 2801)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 2801)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 2783)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 2806)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 2806)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 2806)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 2806)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 2783)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 2783)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

DetailedParameterControl.ToolBar (Page 2678)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

DetailedParameterControl.ToolBar (Page 2678)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 2812)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 2812)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 2812)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 2812)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 2812)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 2812)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

DetailedParameterControl.ToolBar (Page 2678)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 2815)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 2815)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 2815)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 2815)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

DetailedParameterControl.ToolBar (Page 2678)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 2818)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 2818)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 2818)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 2818)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

DetailedParameterControl.ToolBar (Page 2678)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

DetailedParameterControl.ToolBar (Page 2678)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

DetailedParameterControl.ToolBar (Page 2678)

DetailedParameterControl.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DetailedParameterControl.Top
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Visible

Description

The "Visible" property specifies whether the parameter set display is visible.

Type

Bool

Access

Read-write

Syntax

```
DetailedParameterControl.Visible
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DetailedParameterControl.Width
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the parameter set display.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title

- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
DetailedParameterControl.WindowFlags
```

Example

Adapting the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the parameter set display.

Syntax

```
DetailedParameterControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the parameter set control.

Syntax

```
DetailedParameterControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
DetailedParameterControl.PropertyFlashing(propertyName, enable[,
value][, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl_OnActivated()

Description

The "OnActivated" event occurs when a parameter set display receives focus:

- A parameter set display is selected via the configured tab sequence.
- A parameter set display that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
DetailedParameterControl_OnActivated(item)
```

Context

`item`

Type: Object

Parameter set display where the event occurs.

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the parameter set display.

Syntax

```
DetailedParameterControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Parameter set display where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when the parameter set display loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
DetailedParameterControl_OnDeactivated(item)
```

Context

item

Type: Object

Parameter set display where the event occurs.

See also

DetailedParameterControl (Page 2587)

DetailedParameterControl_OnInitialized()

Description

The "OnInitialized" event occurs when a parameter set display has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
DetailedParameterControl_OnInitialized(item)
```

Context

item

Type: Object

Parameter set display where the event occurs.

See also

DetailedParameterControl (Page 2587)

DotNetControlContainer

Description

The "DotNetControlContainer" object represents a container for the display of WPF and WinForms control elements in runtime.

Object type

HmiDotNetControlContainer

Properties

The "DotNetControlContainer" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the display.
- **Enabled**
  Specifies whether the display can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the display.
- **Layer**
  Returns the screen layer in which the display is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the display.
- **Operability**
  Returns whether the display is operable.
- **Parent**
  Returns the higher-level screen object.
- **Properties**
  allows access to the dynamic properties of the control elements in the display.
- **RenderingTemplate**
  Returns the name of the template from which the display was created.
- **RequireExplicitUnlock**
  Returns whether the display is only operable while the associated button is being pressed.
- **ShowFocusVisual**
  Specifies whether the display is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the display.
- **TabIndex**
  Returns the position of the display in the tab sequence.

- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the display is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the display.

Methods

The "DotNetControlContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the display.
- **PropertyFlashing()**
  Configures flashing of a property.

DotNetControlContainer.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
DotNetControlContainer.Authorization
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
DotNetControlContainer.Caption
```

See also

DotNetControlContainer (Page 2829)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

DotNetControlContainer.Caption (Page 2831)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2832)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2832)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2832)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2832)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2832)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2832)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

DotNetControlContainer.Caption (Page 2831)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

DotNetControlContainer.Caption (Page 2831)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

DotNetControlContainer.Caption (Page 2831)

DotNetControlContainer.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
DotNetControlContainer.CaptionColor
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects (CustomControl, SwacComponent, or WidgetType).

Type

String

Access

Read-only

Syntax

```
DotNetControlContainer.ContainedType
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the control.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.

- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
DotNetControlContainer.CurrentQuality
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Enabled

Description

The "Enabled" property specifies whether the display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
DotNetControlContainer.Enabled
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DotNetControlContainer.Height
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Icon

Description

The "Icon" property specifies the icon of the display.

Type

String

Access

Read-write

Syntax

```
DotNetControlContainer.Icon
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Layer

Description

The "Layer" property returns the screen layer in which the control is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
DotNetControlContainer.Layer
```

See also

DotNetControlContainer (Page 2829)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

DotNetControlContainer.Layer (Page 2840)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

DotNetControlContainer.Layer (Page 2840)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

DotNetControlContainer.Layer (Page 2840)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

DotNetControlContainer.Layer (Page 2840)

DotNetControlContainer.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DotNetControlContainer.Left
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
DotNetControlContainer.Margin
```

See also

DotNetControlContainer (Page 2829)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

DotNetControlContainer.Margin (Page 2843)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

DotNetControlContainer.Margin (Page 2843)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

DotNetControlContainer.Margin (Page 2843)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

DotNetControlContainer.Margin (Page 2843)

DotNetControlContainer.Name

Description

The "Name" property returns the name of the control.

Type

String

Access

Read-only

Syntax

```
DotNetControlContainer.Name
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
DotNetControlContainer.Operability
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
DotNetControlContainer.Parent
```

See also

DotNetControlContainer (Page 2829)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

DotNetControlContainer.Properties

Description

The "Properties" property allows access to the dynamic properties of the control elements in the display.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
DotNetControlContainer.Properties
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the control was created.

Type

String

Access

Read-only

Syntax

```
DotNetControlContainer.RenderingTemplate
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
DotNetControlContainer.RequireExplicitUnlock
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
DotNetControlContainer.ShowFocusVisual
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the display.

Type

String

Access

Read-only

Syntax

```
DotNetControlContainer.StyleItemClass
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.TabIndex

Description

The "TabIndex" property returns the position of the display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
DotNetControlContainer.TabIndex
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
DotNetControlContainer.Top
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
DotNetControlContainer.Visible
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
DotNetControlContainer.Width
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the display.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
DotNetControlContainer.WindowFlags
```

Example

Adapting the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the display.

Syntax

```
DotNetControlContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

DotNetControlContainer (Page 2829)

DotNetControlContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
DotNetControlContainer.PropertyFlashing(propertyName, enable[,
value][, alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

DotNetControlContainer (Page 2829)

Ellipse

Description

The "Ellipse" object represents an ellipse in runtime.

Object type

HmiEllipse

Properties

The "Ellipse" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.

- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CenterX**
  Specifies the X coordinate of the center.
- **CenterY**
  Sets the Y coordinate of the center.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the ellipse.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the ellipse can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the ellipse is filled.
- **FillLevel**
  Specifies the fill of the ellipse in percent.
- **Layer**
  Returns the layer of the screen in which the ellipse is located.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the ellipse.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the ellipse is operable.
- **Parent**
  Returns the higher-level screen object.
- **RadiusX**
  Specifies the X radius.
- **RadiusY**
  Specifies the Y radius.
- **RequireExplicitUnlock**
  Returns whether the ellipse is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the ellipse rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.

- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the ellipse is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the ellipse.
- **TabIndex**
  Returns the position of the ellipse in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the ellipse is visible.

Methods

The "Ellipse" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the ellipse.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Ellipse" object has the following events:

- **OnActivated()**
  Occurs when an ellipse receives focus.
- **OnContextTapped()**
  Occurs when an ellipse is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when an ellipse loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the ellipse is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the ellipse is in focus.
- **OnTapped()**
  Occurs when an ellipse is left-clicked or short-touched.

Ellipse.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Ellipse.AlternateBackColor
```

See also

Ellipse (Page 2855)

Ellipse.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Ellipse.AlternateBorderColor
```

See also

Ellipse (Page 2855)

Ellipse.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Ellipse.Authorization
```

See also

Ellipse (Page 2855)

Ellipse.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Ellipse.BackColor
```

See also

Ellipse (Page 2855)

Ellipse.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
Ellipse.BackFillPattern
```

See also

Ellipse (Page 2855)

Ellipse.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Ellipse.BorderColor
```

See also

Ellipse (Page 2855)

Ellipse.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Ellipse.BorderWidth
```

See also

Ellipse (Page 2855)

Ellipse.CenterX

Description

The "CenterX" property specifies the X coordinate of the center.

Type

Int32

Access

Read-write

Syntax

```
Ellipse.CenterX
```

See also

Ellipse (Page 2855)
Ellipse.CenterY (Page 2862)

Ellipse.CenterY

Description

The "CenterY" property specifies the Y coordinate of the center.

Type

Int32

Access

Read-write

Syntax

```
Ellipse.CenterY
```

See also

Ellipse (Page 2855)
Ellipse.CenterX (Page 2862)

Ellipse.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the ellipse.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Ellipse.CurrentQuality
```

See also

Ellipse (Page 2855)

Ellipse.DashType

Description

The "DashType" property specifies the stroke type of the border or line.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted

- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Ellipse.DashType
```

See also

Ellipse (Page 2855)

Ellipse.Enabled

Description

The "Enabled" property specifies whether the ellipse can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Ellipse.Enabled
```

See also

Ellipse (Page 2855)

Ellipse.FillDirection

Description

The "FillDirection" property specifies the direction from which the ellipse is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom
- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
Ellipse.FillDirection
```

See also

Ellipse (Page 2855)

Ellipse.FillLevel

Description

The "FillLevel" property specifies the fill level of the ellipse in percent.

Type

UInt8

Access

Read-write

Syntax

```
Ellipse.FillLevel
```

See also

Ellipse (Page 2855)

Ellipse.Layer

Description

The "Layer" property returns the layer of the screen in which the ellipse is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Ellipse.Layer
```

See also

Ellipse (Page 2855)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Ellipse.Layer (Page 2865)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Ellipse.Layer (Page 2865)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Ellipse.Layer (Page 2865)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Ellipse.Layer (Page 2865)

Ellipse.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Ellipse.Margin
```

See also

Ellipse (Page 2855)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Ellipse.Margin (Page 2868)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Ellipse.Margin (Page 2868)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Ellipse.Margin (Page 2868)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Ellipse.Margin (Page 2868)

Ellipse.Name

Description

The "Name" property returns the name of the ellipse.

Type

String

Access

Read-only

Syntax

```
Ellipse.Name
```

See also

Ellipse (Page 2855)

Ellipse.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Ellipse.Opacity
```

See also

Ellipse (Page 2855)

Ellipse.Operability

Description

The "Operability" property returns whether the ellipse is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Ellipse.Operability
```

See also

Ellipse (Page 2855)

Ellipse.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Ellipse.Parent
```

See also

Ellipse (Page 2855)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Ellipse.RadiusX

Description

The "RadiusX" property specifies the X radius.

Type

UInt32

Access

Read-write

Syntax

```
Ellipse.RadiusX
```

See also

Ellipse (Page 2855)

Ellipse.RadiusY (Page 2873)

Ellipse.RadiusY

Description

The "RadiusY" property specifies the Y radius.

Type

UInt32

Access

Read-write

Syntax

```
Ellipse.RadiusY
```

See also

Ellipse (Page 2855)

Ellipse.RadiusX (Page 2873)

Ellipse.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the ellipse can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Ellipse.RequireExplicitUnlock
```

See also

Ellipse (Page 2855)

Ellipse.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Ellipse.RotationAngle
```

See also

Ellipse (Page 2855)

Ellipse.RotationCenterPlacement (Page 2875)

Ellipse.RotationCenterX (Page 2876)

Ellipse.RotationCenterY (Page 2876)

Ellipse.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the ellipse rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Ellipse.RotationCenterPlacement
```

See also

Ellipse (Page 2855)

Ellipse.RotationAngle (Page 2874)

Ellipse.RotationCenterX (Page 2876)

Ellipse.RotationCenterY (Page 2876)

Ellipse.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Ellipse.RotationCenterX
```

See also

Ellipse (Page 2855)

Ellipse.RotationAngle (Page 2874)

Ellipse.RotationCenterPlacement (Page 2875)

Ellipse.RotationCenterY (Page 2876)

Ellipse.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Ellipse.RotationCenterY
```

See also

Ellipse (Page 2855)
Ellipse.RotationAngle (Page 2874)
Ellipse.RotationCenterPlacement (Page 2875)
Ellipse.RotationCenterX (Page 2876)

Ellipse.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
Ellipse.ShowFillLevel
```

See also

Ellipse (Page 2855)

Ellipse.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the ellipse is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Ellipse.ShowFocusVisual
```

See also

Ellipse (Page 2855)

Ellipse.StyleItemClass

Description

The "StyleItemClass" property returns the style is applied to the ellipse.

Type

String

Access

Read-only

Syntax

```
Ellipse.StyleItemClass
```

See also

Ellipse (Page 2855)

Ellipse.TabIndex

Description

The "TabIndex" property returns the position of the ellipse in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Ellipse.TabIndex
```

See also

Ellipse (Page 2855)

Ellipse.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Ellipse.ToolTipText
```

See also

Ellipse (Page 2855)

Ellipse.Visible

Description

The "Visible" property specifies whether the ellipse is visible.

Type

Bool

Access

Read-write

Syntax

```
Ellipse.Visible
```

See also

Ellipse (Page 2855)

Ellipse.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the ellipse.

Syntax

```
Ellipse.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Ellipse (Page 2855)

Ellipse.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Ellipse.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Ellipse (Page 2855)

Ellipse_OnActivated()

Description

The "OnActivated" event occurs when an ellipse receives focus:

- An ellipse is selected via the configured tab sequence.
- An ellipse that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Ellipse_OnActivated(item)
```

Context

`item`

Type: Object

Ellipse where the event occurs.

See also

Ellipse (Page 2855)

Ellipse_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- An ellipse is right-clicked.
- An ellipse is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Ellipse_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Ellipse where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Ellipse (Page 2855)

Ellipse_OnDeactivated()

Description

The "OnDeactivated" event occurs when the ellipse loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Ellipse_OnDeactivated(item)
```

Context

item

Type: Object

Ellipse where the event occurs.

See also

Ellipse (Page 2855)

Ellipse_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the ellipse is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Ellipse_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Ellipse where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Ellipse (Page 2855)

Ellipse_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the ellipse is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Ellipse_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Ellipse where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Ellipse (Page 2855)

Ellipse_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- An ellipse is left-clicked.
- The <RETURN> or <SPACE> key is pressed when an ellipse has the focus.
- An ellipse is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Ellipse_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Ellipse where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Ellipse (Page 2855)

EllipseSegment

Description

The "EllipseSegment" object represents an ellipse segment in runtime.

Object type

HmiEllipseSegment

Properties

The "EllipseSegment" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AngleRange**
  Specifies the arc angle.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CenterX**
  Specifies the X coordinate of the rotation point.
- **CenterY**
  Specifies the Y coordinate of the rotation point.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the ellipse segment.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the ellipse segment can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the ellipse segment is filled.
- **FillLevel**
  Specifies the fill of the ellipse segment in percent.
- **Layer**
  Returns the layer of the screen in which the ellipse segment is located.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the ellipse segment.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the ellipse segment is operable.

- **Parent**
  Returns the higher-level screen object.
- **RadiusX**
  Specifies the X radius.
- **RadiusY**
  Specifies the Y radius.
- **RequireExplicitUnlock**
  Returns whether the ellipse segment is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the ellipse segment rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the ellipse segment is highlighted when in focus.
- **StartAngle**
  Specifies the angle by which the start point deviates from the zero position (0°).
- **StyleItemClass**
  Returns the style which is applied to the ellipse segment.
- **TabIndex**
  Returns the position of the ellipse segment in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the ellipse segment is visible.

Methods

The "EllipseSegment" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the ellipse segment.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "EllipseSegment" object has the following events:

- **OnActivated()**
  Occurs when an ellipse segment receives focus.
- **OnContextTapped()**
  Occurs when an ellipse segment is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when an ellipse segment loses the focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the ellipse segment is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the ellipse segment is in focus.
- **OnTapped()**
  Occurs when an ellipse segment is left-clicked or short-touched.

EllipseSegment.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
EllipseSegment.AlternateBackColor
```

See also

EllipseSegment (Page 2888)

EllipseSegment.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
EllipseSegment.AlternateBorderColor
```

See also

EllipseSegment (Page 2888)

EllipseSegment.AngleRange

Description

The "AngleRange" property specifies the arc angle clockwise.

Type

Int32

Access

Read-write

Syntax

```
EllipseSegment.AngleRange
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
EllipseSegment.Authorization
```

See also

EllipseSegment (Page 2888)

EllipseSegment.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
EllipseSegment.BackColor
```

See also

EllipseSegment (Page 2888)

EllipseSegment.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent

- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
EllipseSegment.BackFillPattern
```

See also

EllipseSegment (Page 2888)

EllipseSegment.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
EllipseSegment.BorderColor
```

See also

EllipseSegment (Page 2888)

EllipseSegment.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
EllipseSegment.BorderWidth
```

See also

EllipseSegment (Page 2888)

EllipseSegment.CenterX

Description

The "CenterX" property specifies the X coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
EllipseSegment.CenterX
```

See also

EllipseSegment (Page 2888)

EllipseSegment.CenterY

Description

The "CenterY" property specifies the Y coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
EllipseSegment.CenterY
```

See also

EllipseSegment (Page 2888)

EllipseSegment.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the ellipse segment.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
EllipseSegment.CurrentQuality
```

See also

EllipseSegment (Page 2888)

EllipseSegment.DashType

Description

The "DashType" property specifies the stroke type of the border or line.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
EllipseSegment.DashType
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Enabled

Description

The "Enabled" property specifies whether the ellipse segment can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
EllipseSegment.Enabled
```

See also

EllipseSegment (Page 2888)

EllipseSegment.FillDirection

Description

The "FillDirection" property specifies the direction from which the ellipse segment is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom
- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
EllipseSegment.FillDirection
```

See also

EllipseSegment (Page 2888)

EllipseSegment.FillLevel

Description

The "FillLevel" property specifies the fill level of the ellipse segment in percent.

Type

UInt8

Access

Read-write

Syntax

```
EllipseSegment.FillLevel
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Layer

Description

The "Layer" property returns the layer of the screen in which the ellipse segment is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
EllipseSegment.Layer
```

See also

EllipseSegment (Page 2888)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

EllipseSegment.Layer (Page 2899)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

EllipseSegment.Layer (Page 2899)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

EllipseSegment.Layer (Page 2899)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

EllipseSegment.Layer (Page 2899)

EllipseSegment.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
EllipseSegment.Margin
```

See also

EllipseSegment (Page 2888)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

EllipseSegment.Margin (Page 2902)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

EllipseSegment.Margin (Page 2902)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

EllipseSegment.Margin (Page 2902)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

EllipseSegment.Margin (Page 2902)

EllipseSegment.Name

Description

The "Name" property returns the name of the ellipse segment.

Type

String

Access

Read-only

Syntax

```
EllipseSegment.Name
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
EllipseSegment.Opacity
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Operability

Description

The "Operability" property returns whether the ellipse segment is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
EllipseSegment.Operability
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
EllipseSegment.Parent
```

See also

EllipseSegment (Page 2888)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

EllipseSegment.RadiusX

Description

The "RadiusX" property specifies the X radius.

Type

UInt32

Access

Read-write

Syntax

```
EllipseSegment.RadiusX
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RadiusY

Description

The "RadiusY" property specifies the Y radius.

Type

UInt32

Access

Read-write

Syntax

```
EllipseSegment.RadiusY
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the ellipse segment can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
EllipseSegment.RequireExplicitUnlock
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
EllipseSegment.RotationAngle
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RotationCenterPlacement (Page 2908)

EllipseSegment.RotationCenterX (Page 2909)

EllipseSegment.RotationCenterY (Page 2910)

EllipseSegment.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the ellipse segment rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
EllipseSegment.RotationCenterPlacement
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RotationAngle (Page 2908)

EllipseSegment.RotationCenterX (Page 2909)

EllipseSegment.RotationCenterY (Page 2910)

EllipseSegment.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
EllipseSegment.RotationCenterX
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RotationAngle (Page 2908)

EllipseSegment.RotationCenterPlacement (Page 2908)

EllipseSegment.RotationCenterY (Page 2910)

EllipseSegment.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
EllipseSegment.RotationCenterY
```

See also

EllipseSegment (Page 2888)

EllipseSegment.RotationAngle (Page 2908)

EllipseSegment.RotationCenterPlacement (Page 2908)

EllipseSegment.RotationCenterX (Page 2909)

EllipseSegment.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
EllipseSegment.ShowFillLevel
```

See also

EllipseSegment (Page 2888)

EllipseSegment.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the ellipse segment is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
EllipseSegment.ShowFocusVisual
```

See also

EllipseSegment (Page 2888)

EllipseSegment.StartAngle

Description

The "StartAngle" specifies the angle by which the start point deviates from the zero position (0° corresponds to 3 o'clock).

Type

Int32

Access

Read-write

Syntax

```
EllipseSegment.StartAngle
```

See also

EllipseSegment (Page 2888)

EllipseSegment.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the ellipse segment.

Type

String

Access

Read-only

Syntax

```
EllipseSegment.StyleItemClass
```

See also

EllipseSegment (Page 2888)

EllipseSegment.TabIndex

Description

The "TabIndex" property returns the position of the ellipse segment in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
EllipseSegment.TabIndex
```

See also

EllipseSegment (Page 2888)

EllipseSegment.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
EllipseSegment.ToolTipText
```

See also

EllipseSegment (Page 2888)

EllipseSegment.Visible

Description

The "Visible" property specifies whether the ellipse segment is visible.

Type

Bool

Access

Read-write

Syntax

```
EllipseSegment.Visible
```

See also

EllipseSegment (Page 2888)

EllipseSegment.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the ellipse segment.

Syntax

```
EllipseSegment.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

EllipseSegment (Page 2888)

EllipseSegment.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
EllipseSegment.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

EllipseSegment (Page 2888)

EllipseSegment_OnActivated()

Description

The "OnActivated" event occurs when an ellipse segment receives focus:

- An ellipse segment is selected via the configured tab sequence.
- An ellipse segment that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
EllipseSegment_OnActivated(item)
```

Context

`item`

Type: Object

Ellipse segment where the event occurs.

See also

EllipseSegment (Page 2888)

EllipseSegment_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- An ellipse segment is right-clicked.
- An ellipse segment is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
EllipseSegment_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Ellipse segment where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

EllipseSegment (Page 2888)

EllipseSegment_OnDeactivated()

Description

The "OnDeactivated" event occurs when an ellipse segment loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
EllipseSegment_OnDeactivated(item)
```

Context

item

Type: Object

Ellipse segment where the event occurs.

See also

EllipseSegment (Page 2888)

EllipseSegment_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the ellipse segment is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
EllipseSegment_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Ellipse segment where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

EllipseSegment (Page 2888)

EllipseSegment_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the ellipse segment is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
EllipseSegment_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Ellipse segment where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

EllipseSegment (Page 2888)

EllipseSegment_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- An ellipse segment is left-clicked.
- The <RETURN> or <SPACE> key is pressed when an ellipse segment has the focus.
- An ellipse segment is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
EllipseSegment_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Ellipse segment where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

EllipseSegment (Page 2888)

EllipticalArc

Description

The "EllipticalArc" object represents an elliptical arc in runtime.

Object type

HmiEllipticalArc

Properties

The "EllipticalArc" object has the following properties:

- **AlternateLineColor**
  Specifies the second line color which is displayed for line styles such as "Dash".
- **AngleRange**
  Specifies the arc angle.
- **Authorization**
  Returns the operator authorization.
- **CapType**
  Specifies the shape of the line ends.
- **CenterX**
  Specifies the X coordinate of the rotation point.
- **CenterY**
  Specifies the Y coordinate of the rotation point.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the elliptical arc.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the elliptical arc can be operated in runtime.
- **EndType**
  Specifies the type of line end.
- **Layer**
  Returns the layer of the screen in which the elliptical arc is located.
- **LineColor**
  Specifies the line color.
- **LineWidth**
  Specifies the line thickness.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the elliptical arc.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the elliptical arc is operable.
- **Parent**
  Returns the higher-level screen object.
- **RadiusX**
  Specifies the X radius.
- **RadiusY**
  Specifies the Y radius.

- **RequireExplicitUnlock**
  Returns whether the elliptical arc is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the elliptical arc rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the elliptical arc is highlighted when in focus.
- **StartAngle**
  Specifies the angle by which the start point deviates from the zero position (0°).
- **StartType**
  Specifies the type of line start.
- **StyleItemClass**
  Returns the style which is applied to the elliptical arc.
- **TabIndex**
  Returns the position of the elliptical arc in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the elliptical arc is visible.

Methods

The "EllipticalArc" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the elliptical arc.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "EllipticalArc" object has the following events:

- **OnActivated()**
  Occurs when an elliptical arc receives focus.
- **OnContextTapped()**
  Occurs when an elliptical arc is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when an elliptical arc loses focus.

- **OnKeyDown()**
  Occurs when a key is pressed while the elliptical arc is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the elliptical arc is in focus.
- **OnTapped()**
  Occurs when an elliptical arc is left-clicked or short-touched.

EllipticalArc.AlternateLineColor

Description

The "AlternateLineColor" property specifies the second line color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
EllipticalArc.AlternateLineColor
```

See also

EllipticalArc (Page 2922)

EllipticalArc.AngleRange

Description

The "AngleRange" property specifies the arc angle clockwise.

Type

Int32

Access

Read-write

Syntax

```
EllipticalArc.AngleRange
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
EllipticalArc.Authorization
```

See also

EllipticalArc (Page 2922)

EllipticalArc.CapType

Description

The "CapType" property specifies the shape of the line ends.

Type

Int32, HmiCapType

Specifies the line ends:

- Round (0): Round (line extends beyond the line end point with half the line thickness)
- Square (256): Square (line extends beyond the line end point with half the line thickness)
- Flat (512): Justified (line ends at the line end point)

Access

Read-write

Syntax

```
EllipticalArc.CapType
```

See also

EllipticalArc (Page 2922)

EllipticalArc.CenterX

Description

The "CenterX" property specifies the X coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
EllipticalArc.CenterX
```

See also

EllipticalArc (Page 2922)

EllipticalArc.CenterY

Description

The "CenterY" property specifies the Y coordinate of the rotation point.

Type

Int32

Access

Read-write

Syntax

```
EllipticalArc.CenterY
```

See also

EllipticalArc (Page 2922)

EllipticalArc.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the elliptical arc.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
EllipticalArc.CurrentQuality
```

See also

EllipticalArc (Page 2922)

EllipticalArc.DashType

Description

The "DashType" property specifies the stroke type of the border or line.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
EllipticalArc.DashType
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Enabled

Description

The "Enabled" property specifies whether the elliptical arc can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
EllipticalArc.Enabled
```

See also

EllipticalArc (Page 2922)

EllipticalArc.EndType

Description

The "EndType" property specifies the line end type.

Type

Int32, HmiLineEndType

Specifies the line end type:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reverse arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
EllipticalArc.EndType
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Layer

Description

The "Layer" property returns the layer of the screen in which the elliptical arc is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
EllipticalArc.Layer
```

See also

EllipticalArc (Page 2922)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

EllipticalArc.Layer (Page 2930)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

EllipticalArc.Layer (Page 2930)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

EllipticalArc.Layer (Page 2930)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

EllipticalArc.Layer (Page 2930)

EllipticalArc.LineColor

Description

The "LineColor" property specifies the line color.

Type

UInt32

Access

Read-write

Syntax

```
EllipticalArc.LineColor
```

See also

EllipticalArc (Page 2922)

EllipticalArc.LineWidth

Description

The "LineWidth" property specifies the line thickness.

Type

UInt8

Access

Read-write

Syntax

```
EllipticalArc.LineWidth
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
EllipticalArc.Margin
```

See also

EllipticalArc (Page 2922)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

EllipticalArc.Margin (Page 2934)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

EllipticalArc.Margin (Page 2934)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

EllipticalArc.Margin (Page 2934)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

EllipticalArc.Margin (Page 2934)

EllipticalArc.Name

Description

The "Name" property returns the name of the elliptical arc.

Type

String

Access

Read-only

Syntax

```
EllipticalArc.Name
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
EllipticalArc.Opacity
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Operability

Description

The "Operability" property returns whether the elliptical arc is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
EllipticalArc.Operability
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
EllipticalArc.Parent
```

See also

EllipticalArc (Page 2922)
Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

EllipticalArc.RadiusX

Description

The "RadiusX" property specifies the X radius.

Type

UInt32

Access

Read-write

Syntax

```
EllipticalArc.RadiusX
```

See also

EllipticalArc (Page 2922)

EllipticalArc.RadiusY

Description

The "RadiusY" property specifies the Y radius.

Type

UInt32

Access

Read-write

Syntax

```
EllipticalArc.RadiusY
```

See also

EllipticalArc (Page 2922)

EllipticalArc.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the elliptical arc can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
EllipticalArc.RequireExplicitUnlock
```

See also

EllipticalArc (Page 2922)

EllipticalArc.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
EllipticalArc.RotationAngle
```

See also

EllipticalArc (Page 2922)

EllipticalArc.RotationCenterPlacement (Page 2940)

EllipticalArc.RotationCenterX (Page 2941)

EllipticalArc.RotationCenterY (Page 2942)

EllipticalArc.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the elliptical arc rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
EllipticalArc.RotationCenterPlacement
```

See also

EllipticalArc (Page 2922)

EllipticalArc.RotationAngle (Page 2940)

EllipticalArc.RotationCenterX (Page 2941)

EllipticalArc.RotationCenterY (Page 2942)

EllipticalArc.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
EllipticalArc.RotationCenterX
```

See also

EllipticalArc (Page 2922)
EllipticalArc.RotationAngle (Page 2940)
EllipticalArc.RotationCenterPlacement (Page 2940)
EllipticalArc.RotationCenterY (Page 2942)

EllipticalArc.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
EllipticalArc.RotationCenterY
```

See also

EllipticalArc (Page 2922)
EllipticalArc.RotationAngle (Page 2940)
EllipticalArc.RotationCenterPlacement (Page 2940)
EllipticalArc.RotationCenterX (Page 2941)

EllipticalArc.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the elliptical arc is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
EllipticalArc.ShowFocusVisual
```

See also

EllipticalArc (Page 2922)

EllipticalArc.StartAngle

Description

The "StartAngle" specifies the angle by which the start point deviates from the zero position (0° corresponds to 3 o'clock).

Type

Int32

Access

Read-write

Syntax

```
EllipticalArc.StartAngle
```

See also

EllipticalArc (Page 2922)

EllipticalArc.StartType

Description

The "StartType" property specifies the type of line start.

Type

Int32, HmiLineEndType

Specifies the start of the line:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reverse arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
EllipticalArc.StartType
```

See also

EllipticalArc (Page 2922)

EllipticalArc.StyleItemClass

Description

The "StyleItemClass" property returns the style which applied to the elliptical arc.

Type

String

Access

Read-only

Syntax

```
EllipticalArc.StyleItemClass
```

See also

EllipticalArc (Page 2922)

EllipticalArc.TabIndex

Description

The "TabIndex" property returns the position of the elliptical arc in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
EllipticalArc.TabIndex
```

See also

EllipticalArc (Page 2922)

EllipticalArc.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
EllipticalArc.ToolTipText
```

See also

EllipticalArc (Page 2922)

EllipticalArc.Visible

Description

The "Visible" property specifies whether the elliptical arc is visible.

Type

Bool

Access

Read-write

Syntax

```
EllipticalArc.Visible
```

See also

EllipticalArc (Page 2922)

EllipticalArc.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the elliptical arc.

Syntax

```
EllipticalArc.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

EllipticalArc (Page 2922)

EllipticalArc.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
EllipticalArc.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

EllipticalArc (Page 2922)

EllipticalArc_OnActivated()

Description

The "OnActivated" event occurs when an elliptical arc receives focus:

- An elliptical arc is selected via the configured tab sequence.
- An elliptical arc that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
EllipticalArc_OnActivated(item)
```

Context

item

Type: Object

Elliptical arc where the event occurs.

See also

EllipticalArc (Page 2922)

EllipticalArc_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- An elliptical arc is right-clicked.
- An elliptical arc is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
EllipticalArc_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Elliptical arc where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

EllipticalArc (Page 2922)

EllipticalArc_OnDeactivated()

Description

The "OnDeactivated" event occurs when an elliptical arc loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
EllipticalArc_OnDeactivated(item)
```

Context

`item`

Type: Object

Elliptical arc where the event occurs.

See also

EllipticalArc (Page 2922)

EllipticalArc_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the elliptical arc is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
EllipticalArc_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Elliptical arc where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

EllipticalArc (Page 2922)

EllipticalArc_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the elliptical arc is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
EllipticalArc_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Elliptical arc where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

EllipticalArc (Page 2922)

EllipticalArc_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- An elliptical arc is left-clicked.
- The <RETURN> or <SPACE> key is pressed when an elliptical arc has the focus.
- An elliptical arc is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
EllipticalArc_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Elliptical arc where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

EllipticalArc (Page 2922)

FaceplateContainer

Description

The "FaceplateContainer" object represents a container for faceplate instances in runtime.

Object type

HmiFaceplateContainer

Properties

The "FaceplateContainer" object has the following properties:

- **Adaption**
  Specifies whether the faceplate container adapts its size.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the faceplate container.
- **Enabled**
  Specifies whether the faceplate container can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon.
- **Layer**
  Returns the layer of the screen in which the faceplate container is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the faceplate container.
- **Parent**
  Returns the higher-level screen object.
- **Properties**
  Enables access to the interface data of the faceplate container.
- **RenderingTemplate**
  Returns the name of the template from which the faceplate container was created.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the faceplate container rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.

- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the faceplate container is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the faceplate container.
- **TabIndex**
  Returns the position of the faceplate container in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the faceplate container is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the faceplate container.

Methods

The "FaceplateContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the faceplate container.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "FaceplateContainer" object has the following events:

- **OnActivated()**
  Occurs when a faceplate container receives focus.
- **OnDeactivated()**
  Occurs when a faceplate container loses focus.

FaceplateContainer.Adaption

Description

The "Adaption" property specifies whether the faceplate container adapts its size.

Type

Int32, HmiScreenWindowAdaption

Specifies the adaptation:

- None (0): No adaptation.
- WindowToScreen (1): Window size corresponds to screen size.
- ScreenToWindow (2): Screen size is adapted to window size.

Access

Read-write

Syntax

```
FaceplateContainer.Adaption
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
FaceplateContainer.Caption
```

See also

FaceplateContainer (Page 2954)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

FaceplateContainer.Caption (Page 2957)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 2957)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 2957)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 2957)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 2957)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 2957)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 2957)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

FaceplateContainer.Caption (Page 2957)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

FaceplateContainer.Caption (Page 2957)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

FaceplateContainer.Caption (Page 2957)

FaceplateContainer.CaptionColor

Description

The "CaptionColor" property specifies the background color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
FaceplateContainer.CaptionColor
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects.

Type

String

Access

Read-only

Syntax

```
FaceplateContainer.ContainedType
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the faceplate container.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
FaceplateContainer.CurrentQuality
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Enabled

Description

The "Enabled" property specifies whether the faceplate container can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
FaceplateContainer.Enabled
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
FaceplateContainer.Height
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Icon

Description

The "Icon" property specifies the icon.

Type

String

Access

Read-write

Syntax

```
FaceplateContainer.Icon
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Layer

Description

The "Layer" property returns the layer of the screen in which the faceplate container is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
FaceplateContainer.Layer
```

See also

FaceplateContainer (Page 2954)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

FaceplateContainer.Layer (Page 2966)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

FaceplateContainer.Layer (Page 2966)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

FaceplateContainer.Layer (Page 2966)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

FaceplateContainer.Layer (Page 2966)

FaceplateContainer.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
FaceplateContainer.Left
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
FaceplateContainer.Margin
```

See also

FaceplateContainer (Page 2954)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

FaceplateContainer.Margin (Page 2969)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

FaceplateContainer.Margin (Page 2969)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

FaceplateContainer.Margin (Page 2969)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

FaceplateContainer.Margin (Page 2969)

FaceplateContainer.Name

Description

The "Name" property returns the name of the faceplate container.

Type

String

Access

Read-only

Syntax

```
FaceplateContainer.Name
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
FaceplateContainer.Parent
```

See also

FaceplateContainer (Page 2954)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

FaceplateContainer.Properties

Description

The "Properties" property enables access to the interface data (interface tags and interface properties) of the faceplate container.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
FaceplateContainer.Properties
```

Example

Outputting the name of the connected project tag to the "Tag_1" interface tag:

Copy code

```
let TagName;
TagName = Faceplate.Parent.Properties.Tag_1.Tag;
HMIRuntime.Trace("System Tag Name: " + TagName);
```

Outputting the value of the "CustomProperty_1" interface property:

Copy code

```
let PropertyValue;
PropertyValue = Faceplate.Parent.Properties.CustomProperty_1;
HMIRuntime.Trace("Value of CustomProperty_1: " + PropertyValue);
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the faceplate container was created.

Type

String

Access

Read-only

Syntax

```
FaceplateContainer.RenderingTemplate
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
FaceplateContainer.RotationAngle
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.RotationCenterPlacement (Page 2974)

FaceplateContainer.RotationCenterX (Page 2975)

FaceplateContainer.RotationCenterY (Page 2976)

FaceplateContainer.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the faceplate container rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
FaceplateContainer.RotationCenterPlacement
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.RotationAngle (Page 2974)

FaceplateContainer.RotationCenterX (Page 2975)

FaceplateContainer.RotationCenterY (Page 2976)

FaceplateContainer.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
FaceplateContainer.RotationCenterX
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.RotationAngle (Page 2974)

FaceplateContainer.RotationCenterPlacement (Page 2974)

FaceplateContainer.RotationCenterY (Page 2976)

FaceplateContainer.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
FaceplateContainer.RotationCenterY
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.RotationAngle (Page 2974)

FaceplateContainer.RotationCenterPlacement (Page 2974)

FaceplateContainer.RotationCenterX (Page 2975)

FaceplateContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the faceplate container is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
FaceplateContainer.ShowFocusVisual
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the faceplate container.

Type

String

Access

Read-only

Syntax

```
FaceplateContainer.StyleItemClass
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.TabIndex

Description

The "TabIndex" property returns the position of the faceplate container in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
FaceplateContainer.TabIndex
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
FaceplateContainer.Top
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Visible

Description

The "Visible" property specifies whether the faceplate container is visible.

Type

Bool

Access

Read-write

Syntax

```
FaceplateContainer.Visible
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
FaceplateContainer.Width
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the faceplate container.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized

- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
FaceplateContainer.WindowFlags
```

Example

Adapting the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the faceplate container.

Syntax

```
FaceplateContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

FaceplateContainer (Page 2954)

FaceplateContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
FaceplateContainer.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

FaceplateContainer (Page 2954)

FaceplateContainer_OnActivated()

Description

The "OnActivated" event occurs when a faceplate container receives focus:

- A faceplate container is selected via the configured tab sequence.
- A faceplate container that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
FaceplateContainer_OnActivated(item)
```

Context

item

Type: Object

Faceplate container where the event occurs.

See also

FaceplateContainer (Page 2954)

FaceplateContainer_OnDeactivated()

Description

The "OnDeactivated" event occurs when a faceplate container loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
FaceplateContainer_OnDeactivated(item)
```

Context

item

Type: Object

Faceplate container where the event occurs.

See also

FaceplateContainer (Page 2954)

FaceplateContainer_OnInterface_Event()

Description

The "OnInterface_Event" event stands for user-defined interface events of a faceplate type. In the "Faceplate types" editor you can add any number of interface events with parameters to the faceplate type.

A user-defined interface event can only be triggered via the `FaceplateType.RaiseEvent()` method.

Syntax

```
FaceplateContainer_OnInterface_Event(item[, Parameter_1][,...
Parameter_n])
```

Context

`item`

Type: Object

Faceplate container where the event occurs.

`Parameter_1`

Type: Bool | Byte | Char | Color | DateTime | DInt | DWord | HmiEventTrigger | HmiGesture | HmiKeyboardModifier | Int | LInt | LReal | LString | LWord | Real | SInt | String | Time | UDInt | UInt | ULInt | USInt | Word

Parameters of the event. Any number of parameters can be added to a user-defined interface event.

See also

FaceplateContainer (Page 2954)

FaceplateType.RaiseEvent() (Page 3008)

FaceplateType

Description

The "FaceplateType" object represents a faceplate type in runtime.

Object type

HmiFaceplateType

Properties

The "FaceplateType" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BackGraphic**
  Specifies the background graphic of the faceplate type.
- **BackGraphicStretchMode**
  Specifies the type of scaling of the background graphic of the faceplate type.
- **BackgroundFillMode**
  Specifies the fill area of the background fill.

- **DisplayName**
  Specifies the display name of the faceplate type.
- **Enabled**
  Specifies whether the faceplate type can be operated in runtime.
- **EnableExplicitUnlock**
  Returns which button must be pressed for the faceplate type to be operable.
- **Height**
  Specifies the height.
- **HorizontalAlignment**
  Specifies the horizontal alignment.
- **HotKeys**
  Returns the hotkeys specified for the faceplate type.
- **Items**
  Returns a list of all screen objects of the faceplate type.
- **Layers**
  Returns the list of all layers of the faceplate type.
- **Name**
  Returns the name of the faceplate type.
- **Parent**
  Returns the faceplate container which contains the current faceplate instance as a child.
- **Properties**
  Enables access to the interface data of the faceplate type.
- **VerticalAlignment**
  Specifies the vertical alignment.
- **Width**
  Specifies the width.

Methods

The "FaceplateType" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the faceplate type.
- **Close()**
  Hides a faceplate container or closes a faceplate popup window.
- **FindItem()**
  Searches for and references screen windows or screen items through their object path.
- **OpenFaceplateInPopup()**
  Opens a faceplate from a faceplate type in a popup window.
- **PropertyFlashing()**
  Configures flashing of a property.
- **RaiseEvent()**
  Triggers a user-defined interface event of a faceplate type.

Events

The "FaceplateType" object has the following events:

- **OnContextTapped()**
  Occurs when a faceplate type is right-clicked or long-touched.
- **OnHotKey()**
  Occurs when a hotkey is pressed.
- **OnLoaded()**
  Occurs after the faceplate has been built.
- **OnTapped()**
  Occurs when a faceplate type is left-clicked or short-touched.
- **OnUnloaded()**
  Occurs after the faceplate has been cleared.

FaceplateType.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Faceplate.AlternateBackColor
```

See also

FaceplateType (Page 2984)

FaceplateType.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Faceplate.BackColor
```

See also

FaceplateType (Page 2984)

FaceplateType.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent. Depending on the runtime settings, visible objects in the background can be selected.
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
Faceplate.BackFillPattern
```

See also

FaceplateType (Page 2984)

FaceplateType.BackGraphic

Description

The "BackGraphic" property specifies the background graphic of the faceplate type.

Type

String

Access

Read-write

Syntax

```
Faceplate.BackGraphic
```

See also

FaceplateType (Page 2984)

FaceplateType.BackGraphicStretchMode

Description

The "BackGraphicStretchMode" property specifies the type of scaling of the background graphic of the faceplate type.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling:

- None (0): The graphic is shown in its original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Faceplate.BackGraphicStretchMode
```

See also

FaceplateType (Page 2984)

FaceplateType.BackgroundFillMode

Description

The "BackgroundFillMode" property specifies the fill area of the background fill.

Type

Int32, HmiBackgroundFillMode

Specifies the fill area:

- Window (0): The filling is adapted to the size of the window.
- Screen (1): The filling is adapted to the size of the screen.

Access

Read-write

Syntax

```
Faceplate.BackgroundFillMode
```

See also

FaceplateType (Page 2984)

FaceplateType.DisplayName

Description

The "DisplayName" property specifies the display name of the faceplate type.

Type

String

Access

Read-write

Syntax

```
Faceplate.DisplayName
```

See also

FaceplateType (Page 2984)

FaceplateType.Enabled

Description

The "Enabled" property specifies whether the faceplate type can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Faceplate.Enabled
```

See also

FaceplateType (Page 2984)

FaceplateType.EnableExplicitUnlock

Description

The "EnableExplicitUnlock" property returns which button must be pressed for the faceplate type to be operable.

Type

Object, HmiButton (Page 1961)

Access

Read-only

Syntax

```
Faceplate.EnableExplicitUnlock
```

See also

FaceplateType (Page 2984)

Button (Page 1961)

Button

Description

Button (Page 1961)

FaceplateType.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Faceplate.Height
```

See also

FaceplateType (Page 2984)

FaceplateType.HorizontalAlignment

Description

The "HorizontalAlignment" property specifies the horizontal alignment.

Type

Int32, HmiHorizontalAlignment

Specifies the alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
Faceplate.HorizontalAlignment
```

See also

FaceplateType (Page 2984)

FaceplateType.Items

Description

The "Items" property returns a list of all the screen objects of the faceplate type.

Type

Object, HMIScreenItems (Page 2993)

Access

Read-only

Syntax

```
Faceplate.Items
```

See also

FaceplateType (Page 2984)

ScreenItems (Page 2993)

ScreenItems

Description

The "ScreenItems" object is a list of all screen objects ("HmiScreenItemBase" objects) of the screen.

You reference a "ScreenItems" object via the `Screen.Items` property.

Use

The "ScreenItems" object is a list and can be enumerated. You can access the "ScreenItems" list using the index or the tag name.

Object type

HMIScreenItems

Properties

--

Methods

The "ScreenItems" object has the following methods:

- **Item()**
  Returns a screen object of the "ScreenItems" list.

See also

FaceplateType.Items (Page 2992)

ScreenItems.Item()

Description

The "Item" method returns a screen object of a screen window via the "ScreenItems" list.

Syntax

```
ScreenItems[.Item](ScreenItemName);
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "ScreenItems" object.

Parameters

`ScreenItemName`

Type: String, HmiScreenItemBase

Name of the screen object

Return value

Object, HmiScreenItemBase (Page 1571)

See also

ScreenItems (Page 2993)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

FaceplateType.Layers

Description

The "Layers" property returns the list of all layers ("HmiLayerPart" objects) of the faceplate type.

Type

Object, HmiLayerCollection (Page 2995)

Access

Read-only

Syntax

```
Faceplate.Layers
```

See also

FaceplateType (Page 2984)

HmiLayerCollection (Page 2995)

HmiLayerCollection

Description

The "HmiLayerCollection" object is a list of all layers ("Layer" objects) of the screen.

You reference a "HmiLayerCollection" object via the `Screen.Layers` property.

Use

The "HmiLayerCollection" object is a list and can be counted and enumerated. You can access the "HmiLayerCollection" list using the index or the tag name.

Object type

HmiLayerCollection

Properties

The "HmiLayerCollection" object has the following properties:

- **Count**
  Returns the number of layers of the "HmiLayerCollection" list.

Methods

The "HmiLayerCollection" object has the following methods:

- **Item()**
  Returns a level of the "HmiLayerCollection" list.

See also

FaceplateType.Layers (Page 2994)

HmiLayerCollection.Count

Description

The "Count" property returns the number of screen layers of the "HmiLayerCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiLayerCollection.Count
```

See also

HmiLayerCollection (Page 2995)

HmiLayerCollection.Item()

Description

The "Item" method returns a screen layer of the "HmiLayerCollection" list.

Syntax

```
HmiLayerCollection[.Item](HmiLayerName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiLayerCollection" object.

Parameters

`HmiLayerName`

Type: String, HmiLayerPart

Name of the screen layer

Return value

Object, HmiLayerPart (Page 2997)

See also

HmiLayerCollection (Page 2995)

Layer (Page 2997)

Layer

Description

The "Layer" object represents the layers of an object (e.g. screen).

Object type

HmiLayerPart

Properties

The "Layer" object has the following properties:

- **MaximumZoom**
  Specifies the maximum zoom of the screen up to which the level is to be seen.
- **MinimumZoom**
  Specifies the minimum zoom of the screen up to which the level is to be seen.
- **Name**
  Returns the name of the screen layer.
- **Visible**
  Specifies whether the screen layer and contained objects are visible.

Methods

--

See also

FaceplateType.Layers (Page 2994)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Layer (Page 2997)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Layer (Page 2997)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Layer (Page 2997)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Layer (Page 2997)

FaceplateType.Name

Description

The "Name" property returns the name of the faceplate type.

Type

String

Access

Read-only

Syntax

```
Faceplate.Name
```

See also

FaceplateType (Page 2984)

FaceplateType.Parent

Description

The "Parent" property returns the faceplate container which contains the current faceplate instance as a child.

Type

Object, HmiFaceplateContainer (Page 2954)

Access

Read-only

Syntax

```
Faceplate.Parent
```

See also

FaceplateType (Page 2984)

FaceplateContainer (Page 2954)

FaceplateContainer

Description

FaceplateContainer (Page 2954)

FaceplateType.Properties

Description

The "Properties" property enables access to the interface data (interface tags and interface properties) of the faceplate type.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
Faceplate.Properties
```

Example

Outputting the name of the connected project tag to the "Tag_1" interface tag:

Copy code

```
let TagName;
TagName = Faceplate.Properties.Tag_1.Tag;
HMIRuntime.Trace("System Tag Name: " + TagName);
```

Outputting the value of the "CustomProperty_1" interface property:

Copy code

```
let PropertyValue;
PropertyValue = Faceplate.Properties.CustomProperty_1;
HMIRuntime.Trace("Value of CustomProperty_1: " + PropertyValue);
```

See also

FaceplateType (Page 2984)

FaceplateType.VerticalAlignment

Description

The "VerticalAlignment" property specifies the vertical alignment.

Type

Int32, HmiVerticalAlignment

Specifies the alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
Faceplate.VerticalAlignment
```

See also

FaceplateType (Page 2984)

FaceplateType.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Faceplate.Width
```

See also

FaceplateType (Page 2984)

FaceplateType.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the faceplate type.

Syntax

```
Faceplate.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

FaceplateType (Page 2984)

FaceplateType.Close()

Description

The "Close" method hides a faceplate container or closes a faceplate popup window.

Syntax

```
Faceplate.Close()
```

Parameters

--

Return value

Bool

See also

FaceplateType (Page 2984)

FaceplateType.FindItem()

Description

The "FindItem" method searches for and references screen windows or screen objects through their object path.

Syntax

```
Faceplate.FindItem(screenItemPath)
```

Parameters

`ScreenItemPath`

Type: String

Object path of the screen window or screen object you are looking for.

Note

The "Faceplate.FindItem" method has the faceplate in which the script is running as the search context and can also use relative object paths.

Formulation of the object path

The syntax of the object path orients itself to the notation of the file system paths. The object path consists of the names of the screen windows (Screen Windows) and screen objects (Screen Items). The names are connected via a slash ("/") according to the hierarchical positioning. Screens (Screens) and their names are not used in the formulation.

Relative and absolute object paths are distinguished by the prefix of the object path. The following prefixes can be used:

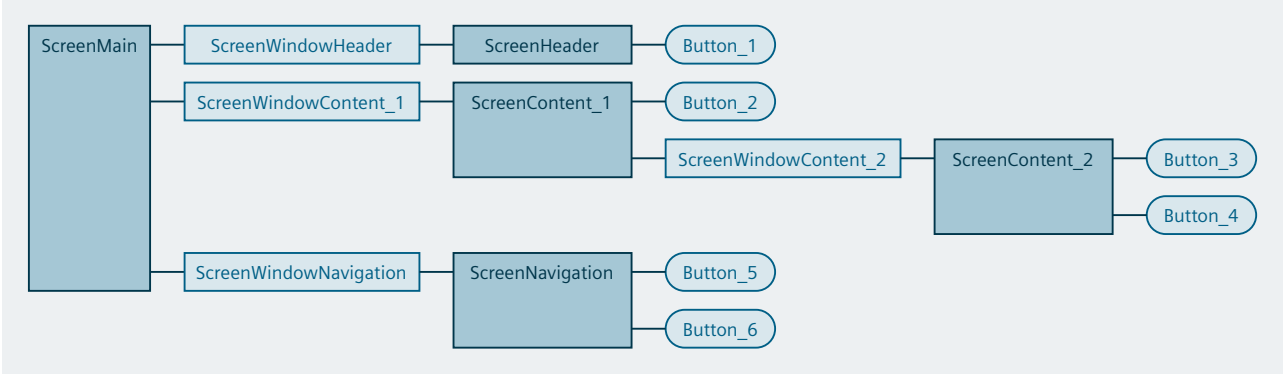
- Relative object path
  - "..": References the higher-level screen window (parent) in the context of the current screen window.
  - ".": References own screen window (self).
  - "": Without a prefix, a screen object of the current screen window is referenced.
- Absolute object paths
  - "/": References a screen window on the highest level, whose name must follow.
  - "~": References the screen window on the highest level in own screen hierarchy.

Additional rules for formulating an object path:

- The ".." string may be used multiple times in the object path, but only together at the beginning of the object path, for example, "../../Window5".
- If the object path does not end with a screen object name, a screen window is referenced.
- A search is performed for screen objects of the object path in the screens of the referenced screen window. It is not permitted to specify a screen name.

Examples of object paths

The following window-screen object hierarchy is adopted for the examples:



This results in the following object paths for addressing the objects:

- Relative addressing:
  - "Button_2" changes the label of "Button_1":
    ```
    // Navigate one level up and find "Button_1" inside the
    "ScreenWindowHeader"
    Screen.FindItem("../ScreenWindowHeader/Button_1").Text =
    "Changed by Button_2"
    ```
  - "Button_3" changes the label of "Button_5":
    ```
    // Navigate two levels up and find "Button 2" inside the
    "ScreenWindowNavigation"
    Screen.FindItem("../../ScreenWindowNavigation/Button_5").Text =
    "Changed by Button_3"
    ```
  - "Button_3" changes the label of "Button_4":
    ```
    // Find "Button_4" in same screen ("ScreenContent_2")
    Screen.FindItem("Button_4").Text = "Changed by Button_3"
    Screen.FindItem("./Button_4").Text = "Changed by Button_3"
    ```
- Absolute addressing:
  - "Button_4" changes the label of "Button_6":
    ```
    // Navigate to the root screen and find "Button_6" inside the
    "ScreenWindowNavigation"
    Screen.FindItem("~/ScreenWindowNavigation/Button_6").Text =
    "Changed by Button_4"
    ```

Return value

Object, HmiScreenObjectBase (Page 1571)

See also

FaceplateType (Page 2984)

Screen Items (Page 1571)

FaceplateType.OpenFaceplateInPopup()

Description

The "OpenFaceplateInPopup" method opens a faceplate from a faceplate type in a popup window.

Syntax

```
Faceplate.OpenFaceplateInPopup(faceplateType, title[,
independentWindow][, invisible])
```

Parameters

faceplateType

Type: String, HmiFaceplateType

Name of the faceplate type

title

Type: String

Title of the popup window

independentWindow

Optional, type: Bool

Specifies the dependency of the popup window on the calling faceplate:

- True: The popup window remains open until either the popup window is closed manually or runtime is exited.
- False: The popup window is closed when the faceplate or the screen is exited.

invisible

Optional, type: Bool

Causes the faceplate to be configured so that it is invisible to the operator.

To then display the faceplate, set the property `visible=true`.

Return value

Object, HmiPopupScreenWindow (Page 4288)

See also

FaceplateType (Page 2984)

PopupScreenWindow (Page 4288)

FaceplateType.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Faceplate.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

FaceplateType (Page 2984)

FaceplateType.RaiseEvent()

Description

The "RaiseEvent" method triggers a user-defined interface event of a faceplate type.

Syntax

```
Faceplate.RaiseEvent(name, parameters)
```

Parameters

`name`

Type: String

Name of the interface event

`parameters`

Type: Object

Parameter of the interface event

Return value

Bool

See also

FaceplateType (Page 2984)

FaceplateContainer_OnInterface_Event() (Page 2983)

FaceplateType_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A faceplate type is right-clicked.
- A faceplate type is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
FaceplateType_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Faceplate type where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

FaceplateType (Page 2984)

FaceplateType_OnHotKey()

Description

The "OnHotKey" event occurs when the operator presses a hotkey:

Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Syntax

```
FaceplateType_OnHotKey(item,keyCode,modifiers)
```

Context

item

Type: Object

Faceplate type where the event occurs.

keyCode

Type: DInt

Numeric identifier of the pressed hotkey

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

FaceplateType (Page 2984)

FaceplateType_OnLoaded()

Description

The "OnLoaded" event occurs after the faceplate type has been built.

Syntax

```
FaceplateType_OnLoaded(item)
```

Context

item

Type: Object

Faceplate type where the event occurs.

See also

FaceplateType (Page 2984)

FaceplateType_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A faceplate type is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a faceplate type has the focus.
- A faceplate type is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
FaceplateType_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Faceplate type where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

FaceplateType (Page 2984)

FaceplateType_OnUnloaded()

Description

The "OnUnloaded" event occurs after the faceplate type has been cleared.

Syntax

```
FaceplateType_OnUnloaded(item)
```

Context

item

Type: Object

Faceplate type where the event occurs.

See also

FaceplateType (Page 2984)

PopupScreenWindow

Description

PopupScreenWindow (Page 4288)

FunctionTrendControl

Description

The "FunctionTrendControl" object represents an f(x) trend view of tag values as a function of a different tag in runtime.

Object type

HmiFunctionTrendControl

Properties

The "FunctionTrendControl" object has the following properties:

- **AreaSpacing**
  Specifies the distance between function trend areas.
- **BackColor**
  Specifies the background color.

- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the f(x) trend view.
- **Enabled**
  Specifies whether the f(x) trend view can be operated in runtime.
- **ExtendRulerToAxis**
  Specifies whether the ruler is extended into the axis.
- **FunctionTrendAreas**
  Returns the function trend areas of the f(x) trend view.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the f(x) trend view.
- **Layer**
  Returns the screen layer in which the f(x) trend view is located.
- **Left**
  Specifies the value of the X coordinate.
- **Legend**
  Specifies the appearance of the legend.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the f(x) trend view.
- **Online**
  Specifies the start and stop of the update of the f(x) trend view.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the f(x) trend view.was created.
- **ShiftAxis**
  Specifies whether to swap the X axis and Y axis of the f(x) trend view.
- **ShowFocusVisual**
  Specifies whether the f(x) trend view is highlighted when in focus.
- **ShowRuler**
  Specifies whether the ruler is displayed for determining a function trend value.
- **ShowStatisticRulers**
  Specifies whether to display the two rulers for specifying the statistics area.
- **StatusBar**
  Specifies the information bar of the f(x) trend view.

- **StyleItemClass**
  Returns the style which is applied to the f(x) trend view.
- **TabIndex**
  Returns the position of the f(x) trend view in the tab sequence.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Specifies the toolbar of the f(x) trend view.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the f(x) trend view is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the f(x) trend view.

Methods

The "FunctionTrendControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the f(x) trend view.
- **FireCommand()**
  Executes the command of an element of the toolbar or information bar of the f(x) trend view.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "FunctionTrendControl" object has the following events:

- **OnActivated()**
  Occurs when a f(x) trend view receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the f(x) trend view.
- **OnDeactivated()**
  Occurs when a f(x) trend view loses focus.
- **OnInitialized()**
  Occurs when a f(x) trend view has been successfully initialized and the data connection to the PLC has been established.

FunctionTrendControl.AreaSpacing

Description

The "AreaSpacing" property specifies the spacing between function trend areas.

Type

UInt16

Access

Read-write

Syntax

```
FunctionTrendControl.AreaSpacing
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendControl.BackColor
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
FunctionTrendControl.Caption
```

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

FunctionTrendControl.Caption (Page 3018)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 3018)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 3018)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 3018)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 3018)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 3018)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 3018)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

FunctionTrendControl.Caption (Page 3018)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

FunctionTrendControl.Caption (Page 3018)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

FunctionTrendControl.Caption (Page 3018)

FunctionTrendControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendControl.CaptionColor
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the f(x) trend view.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
FunctionTrendControl.CurrentQuality
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Enabled

Description

The "Enabled" property specifies whether the f(x) trend view can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.Enabled
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.ExtendRulerToAxis

Description

The "ExtendRulerToAxis" property specifies whether the ruler is extended to the axis.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.ExtendRulerToAxis
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.FunctionTrendAreas

Description

The "FunctionTrendAreas" property returns the function trend areas ("FunctionTrendArea" objects) of the f(x) trend view.

Type

Object, HmiFunctionTrendAreaCollection (Page 3026)

Access

Read-only

Syntax

```
FunctionTrendControl.TrendAreas
```

See also

FunctionTrendControl (Page 3014)

HmiFunctionTrendAreaCollection (Page 3026)

HmiFunctionTrendAreaCollection

Description

The "HmiFunctionTrendAreaCollection" object is a list of all function trend areas ("FunctionTrendArea" objects) of the f(x) trend view.

Use

The "HmiFunctionTrendAreaCollection" object is a list and can be counted and enumerated. You can access the "HmiFunctionTrendAreaCollection" list using the index or the tag names.

Object type

HmiFunctionTrendAreaCollection

Properties

The "HmiFunctionTrendAreaCollection" object has the following properties:

- **Count**
  Returns the number of function trend areas of the "HmiFunctionTrendAreaCollection" list.

Methods

The "HmiFunctionTrendAreaCollection" object has the following methods:

- **Item()**
  Returns a function trend area of the "HmiFunctionTrendAreaCollection" list.

See also

FunctionTrendControl.FunctionTrendAreas (Page 3025)

HmiFunctionTrendAreaCollection.Count

Description

The "Count" property returns the number of trend areas in the "HmiFunctionTrendAreaCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiFunctionTrendAreaCollection.Count
```

See also

HmiFunctionTrendAreaCollection (Page 3026)

HmiFunctionTrendAreaCollection.Item()

Description

The "Item" method returns a function trend area of the "HmiFunctionTrendAreaCollection" list.

Syntax

```
HmiFunctionTrendAreaCollection[.Item](HmiFunctionTrendAreaName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiFunctionTrendAreaCollection" object.

Parameters

`HmiFunctionTrendAreaName`

Type: String

Name of the function trend area

Return value

Object, HmiFunctionTrendAreaPart (Page 3026)

See also

HmiFunctionTrendAreaCollection (Page 3026)

FunctionTrendArea (Page 3028)

FunctionTrendArea

Description

The "TrendArea" object represents a function trend area of the f(x) trend view.

Object type

HmiFunctionTrendAreaPart

Properties

The "FunctionTrendArea" object has the following properties:

- **BackColor**
  Specifies the background color.
- **BottomValueAxes**
  Returns the lower value axes of the function trend area.
- **FunctionTrends**
  Returns the function trends of the function trend area.
- **GridLines**
  Specifies the grid lines of the function trend area.
- **LeftValueAxes**
  Returns the left value axes of the function trend area.
- **MajorGridLinesColor**
  Specifies the color of the main grid lines.
- **MinorGridLinesColor**
  Specifies the color of the auxiliary grid lines.
- **Name**
  Returns the name of the function trend area.
- **RightValueAxes**
  Returns the right value axes of the function trend area.

- **Ruler**
  Specifies the appearance of the ruler to determine the function trend value.
- **SelectedFunctionTrend**
  Specifies the selected trend of the function trend area.
- **SizeFactor**
  Specifies the scaling factor of the function trend area relative to its height.
- **TopValueAxes**
  Returns the upper value axes of the function trend area.
- **Visible**
  Specifies whether the function trend area is visible.

Methods

--

See also

HmiFunctionTrendAreaCollection (Page 3026)

HmiFunctionTrendAreaCollection.Item() (Page 3027)

FunctionTrendArea.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendArea.BackColor
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendArea.BottomValueAxes

Description

The "BottomValueAxes" property returns the lower value axes of the function trend area.

Type

Object, HmiXValueAxisCollection (Page 3030)

Access

Read-only

Syntax

```
FunctionTrendArea.BottomValueAxes
```

See also

FunctionTrendArea (Page 3028)

HmiXValueAxisCollection (Page 3030)

HmiXValueAxisCollection

Description

The "HmiXValueAxisCollection" object is a list of all value axes ("XValueAxis" objects) of the function trend area.

Use

The "HmiXValueAxisCollection" object is a list and can be counted and enumerated. You can access the "HmiXValueAxisCollection" list using the index or the tag names.

Object type

HmiXValueAxisCollection

Properties

The "HmiXValueAxisCollection" object has the following properties:

- **Count**
  Returns the number of value axes in the "HmiXValueAxisCollection" list.

Methods

The "HmiXValueAxisCollection" object has the following methods:

- **Item()**
  Returns a value axis of the "HmiXValueAxisCollection" list.

See also

FunctionTrendArea.BottomValueAxes (Page 3030)

HmiXValueAxisCollection.Count

Description

The "Count" property returns the number of value axes in the "HmiXValueAxisCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiXValueAxisCollection.Count
```

See also

HmiXValueAxisCollection (Page 3030)

HmiXValueAxisCollection.Item()

Description

The "Item" method returns a value axis of the "HmiXValueAxisCollection" list.

Syntax

```
HmiXValueAxisCollection[.Item](HmiXValueAxisName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiXValueAxisCollection" object.

Parameters

`HmiXValueAxisName`

Type: String

Name of the value axis

Return value

Object, HmiXValueAxisPart (Page 3032)

See also

HmiXValueAxisCollection (Page 3030)

XValueAxis (Page 3032)

XValueAxis

Description

The "XValueAxis" object represents a value axis of the function trend area.

Object type

HmiXValueAxisPart

Properties

The "XValueAxis" object has the following properties:

- **ApplyScalingEntries**
  Specifies whether the user scaling of the axis section is applied.
- **AutoRange**
  Specifies whether automatic determination of the value range is activated by the minimum and maximum value of the function trend.
- **AutoScaling**
  Specifies whether the automatic scaling is activated.
- **AxisColor**
  Specifies the color of the value axis.
- **BeginValue**
  Specifies the start of a value range or value range section.
- **DisplayName**
  Specifies the display name of the value axis.
- **DivisionCount**
  Specifies the number of main units with subdivisions.

- **EndValue**
  Specifies the end of a value range or value range section.
- **HelpLines**
  Returns the appearance of the help lines.
- **LabelColor**
  Specifies the color of the axis labeling.
- **LabelFont**
  Specifies the font of the axis labeling.
- **LargeTickLabelingStep**
  Specifies the interval at which scale sections are labeled.
- **MeasurementUnit**
  Returns the displayed unit.
- **MeasurementUnitType**
  Specifies the display format of the unit.
- **OutputFormat**
  Specifies the format for displaying the axis values.
- **ScalingEntries**
  Returns the specification of the user scaling of the axis sections.
- **ScaleMode**
  Specifies the scale mode:
- **ScalingType**
  Specifies the scaling.
- **ShowScalingDisplayNames**
  Specifies whether the display names of the user scaling are used.
- **SubDivisionCount**
  Specifies the number of divisions of the main units.
- **TickColor**
  Specifies the color of the tick marks.
- **Visible**
  Specifies whether the value axis is visible.

Methods

--

See also

HmiXValueAxisCollection (Page 3030)

HmiXValueAxisCollection.Item() (Page 3031)

XValueAxis.ApplyScalingEntries

Description

The "ApplyScalingEntries" property specifies whether the user scaling of the axis sections is applied.

Type

Bool

Access

Read-write

Syntax

```
XValueAxis.ApplyScalingEntries
```

See also

XValueAxis (Page 3032)

XValueAxis.AutoRange

Description

The "AutoRange" property specifies whether automatic determination of the value range is activated by the minimum and maximum value of the function trend.

Type

Bool

Access

Read-write

Syntax

```
XValueAxis.AutoRange
```

See also

XValueAxis (Page 3032)

XValueAxis.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
XValueAxis.AutoScaling
```

See also

XValueAxis (Page 3032)

XValueAxis.AxisColor

Description

The "AxisColor" property specifies the color of the value axis.

Type

UInt32

Access

Read-write

Syntax

```
XValueAxis.BackColor
```

See also

XValueAxis (Page 3032)

XValueAxis.BeginValue

Description

The "BeginValue" property specifies the start of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
XValueAxis.BeginValue
```

See also

XValueAxis (Page 3032)

XValueAxis.DisplayName

Description

The "DisplayName" property specifies the display name of the value axis.

Type

String

Access

Read-write

Syntax

```
XValueAxis.DisplayName
```

See also

XValueAxis (Page 3032)

XValueAxis.DivisionCount

Description

The "DivisionCount" property specifies the number of main units with subdivisions. To this purpose the automatic scaling must be switched off.

Type

Int32

Access

Read-write

Syntax

```
XValueAxis.DivisionCount
```

See also

XValueAxis (Page 3032)

XValueAxis.EndValue

Description

The "EndValue" property specifies the end of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
XValueAxis.EndValue
```

See also

XValueAxis (Page 3032)

XValueAxis.HelpLines

Description

The "HelpLines" property returns the appearance of the help lines.

Type

Object, HmiHelpLineCollection (Page 3038)

Access

Read-only

Syntax

```
XValueAxis.HelpLines
```

See also

XValueAxis (Page 3032)

HmiHelpLineCollection (Page 3038)

HmiHelpLineCollection

Description

The "HmiHelpLineCollection" object is a list of all help lines ("HelpLine" objects).

Use

The "HmiHelpLineCollection" object is a list and can be counted and enumerated. You can access the "HmiHelpLineCollection" list using the index or the tag name.

Object type

HmiHelpLineCollection

Properties

The "HmiHelpLineCollection" object has the following properties:

- **Count**
  Returns the number of help lines of the "HmiHelpLineCollection" list.

Methods

The "HmiHelpLineCollection" object has the following methods:

- **Item()**
  Returns a help line of the "HmiHelpLineCollection" list.

See also

XValueAxis.HelpLines (Page 3038)

HmiHelpLineCollection.Count

Description

The "Count" property returns the number of help lines in the "HmiHelpLineCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiHelpLineCollection.Count
```

See also

HmiHelpLineCollection (Page 3038)

HmiHelpLineCollection.Item()

Description

The "Item" method returns help line of the "HmiHelpLineCollection" list.

Syntax

```
HmiHelpLineCollection[.Item](HmiHelpLineName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiHelpLineCollection" object.

Parameters

HmiHelpLineName

Type: String

Name of the help line

Return value

Object, HmiHelpLinePart (Page 3040)

See also

HmiHelpLineCollection (Page 3038)

HelpLine (Page 3040)

HelpLine

Description

The "HelpLine" object represents a help line of the value axis.

Object type

HmiHelpLinePart

Properties

The "HelpLine" object has the following properties:

- **Value**
  Specifies the value of the help line.
- **Visible**
  Specifies whether the help line is visible.

Methods

--

See also

HmiHelpLineCollection (Page 3038)

HmiHelpLineCollection.Item() (Page 3039)

HelpLine.Value

Description

The "Value" property sets the value of the help line.

Type

Float

Access

Read-write

Syntax

```
HelpLine.Value
```

See also

HelpLine (Page 3040)

HelpLine.Visible

Description

The "Visible" property specifies whether the help line is visible.

Type

Bool

Access

Read-write

Syntax

```
HelpLine.Visible
```

See also

HelpLine (Page 3040)

XValueAxis.LabelColor

Description

The "LabelColor" property specifies the color of the axis labeling.

Type

UInt32

Access

Read-write

Syntax

```
XValueAxis.LabelColor
```

See also

XValueAxis (Page 3032)

XValueAxis.LabelFont

Description

The "LabelFont" property specifies the font of the axis labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
XValueAxis.LabelFont
```

See also

XValueAxis (Page 3032)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

XValueAxis.LabelFont (Page 3042)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

XValueAxis.LabelFont (Page 3042)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

XValueAxis.LabelFont (Page 3042)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

XValueAxis.LabelFont (Page 3042)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

XValueAxis.LabelFont (Page 3042)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

XValueAxis.LabelFont (Page 3042)

XValueAxis.LargeTickLabelingStep

Description

The "LargeTickLabelingStep" property specifies the interval at which scale sections are labeled.

Type

UInt8

Access

Read-write

Syntax

```
XValueAxis.LargeTickLabelingStep
```

See also

XValueAxis (Page 3032)

XValueAxis.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
XValueAxis.MeasurementUnit
```

See also

XValueAxis (Page 3032)

XValueAxis.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
XValueAxis.MeasurementUnitType
```

See also

XValueAxis (Page 3032)

XValueAxis.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the axis values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
XValueAxis.OutputFormat
```

See also

XValueAxis (Page 3032)

XValueAxis.ScalingEntries

Description

The "ScalingEntries" property returns the user scaling of the axes sections.

Type

Object, HmiScalingEntryCollection (Page 3048)

Access

Read-only

Syntax

```
XValueAxis.ScalingEntries
```

See also

XValueAxis (Page 3032)

HmiScalingEntryCollection (Page 3048)

HmiScalingEntryCollection

Description

The "HmiScalingEntryCollection" object is a list of all user-defined axis sections ("ScalingEntry" objects).

Use

The "HmiScalingEntryCollection" object is a list and can be counted and enumerated. You can access the "HmiScalingEntryCollection" list using the index or the tag name.

Object type

HmiHelpLineCollection

Properties

The "HmiScalingEntryCollection" object has the following properties:

- **Count**
  Returns the number of user-defined axis sections of the "HmiScalingEntryCollection" list.

Methods

The "HmiScalingEntryCollection" object has the following methods:

- **Item()**
  Returns a user-defined axis section of the "HmiScalingEntryCollection" list.

See also

XValueAxis.ScalingEntries (Page 3048)

HmiScalingEntryCollection.Count

Description

The "Count" property returns the number of the user-defined axis sections in the "HmiScalingEntryCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiScalingEntryCollection.Count
```

See also

HmiScalingEntryCollection (Page 3048)

HmiScalingEntryCollection.Item()

Description

The "Item" method returns a user-defined axis section of the "HmiScalingEntryCollection" list.

Syntax

```
HmiScalingEntryCollection[.Item](HmiScalingEntryName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiScalingEntryCollection" object.

Parameters

`HmiScalingEntryName`

Type: String

Name of the user-defined axis section

Return value

Object, HmiScalingEntryPart (Page 3050)

See also

HmiScalingEntryCollection (Page 3048)

Scaling Entry (Page 3050)

Scaling Entry

Description

The "ScalingEntry" object represents a user-defined axis section of the value axis.

Object type

HmiScalingEntryPart

Properties

The "YValueAxis" object has the following properties:

- **BeginValue**
  Specifies the start of a value range section.
- **BeginValueTarget**
  Specifies the scaled value for the start of a value range section.
- **DisplayName**
  Specifies the display name of an axis section.
- **EndValue**
  Specifies the end of a value range section.
- **EndValueTarget**
  Specifies the scaled value for the end of a value range section.

Methods

--

See also

HmiScalingEntryCollection (Page 3048)

HmiScalingEntryCollection.Item() (Page 3050)

ScalingEntry.BeginValue

Description

The "BeginValue" property specifies the start of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.BeginValue
```

See also

Scaling Entry (Page 3050)

ScalingEntry.BeginValueTarget

Description

The"BeginValueTarget" property specifies the scaled value for the start of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.BeginValueTarget
```

See also

Scaling Entry (Page 3050)

ScalingEntry.DisplayName

Description

The "DisplayName" property specifies the display name of an axis section.

Type

String

Access

Read-write

Syntax

```
ScalingEntry.DisplayName
```

See also

Scaling Entry (Page 3050)

ScalingEntry.EndValue

Description

The "EndValue" property specifies the end of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.EndValue
```

See also

Scaling Entry (Page 3050)

ScalingEntry.EndValueTarget

Description

The"EndValueTarget" property specifies the scaled value for the end of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.EndValueTarget
```

See also

Scaling Entry (Page 3050)

XValueAxis.ScaleMode

Description

The "ScaleMode" property specifies the scale mode.

Type

Int32, HmiScaleMode

Specifies the scale mode:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
XValueAxis.ScaleMode
```

See also

XValueAxis (Page 3032)

XValueAxis.ScalingType

Description

The "ScalingType" property specifies the scaling.

Type

Int32, HmiScalingType

Specifies the scaling:

- Linear (0): Linear
- Logarithmic (1): Logarithmic
- NegativeLogarithmic (2): Negative logarithmic
- Tangent (4): Tangential
- Quadratic (5): Square
- Cubic (6): Cubic

Access

Read-write

Syntax

```
XValueAxis.ScalingType
```

See also

XValueAxis (Page 3032)

XValueAxis.ShowScalingDisplayNames

Description

The "ShowScalingDisplayNames" property specifies whether the display names of the user scaling are used.

Type

Bool

Access

Read-write

Syntax

```
XValueAxis.ShowScalingDisplayNames
```

See also

XValueAxis (Page 3032)

XValueAxis.SubDivisionCount

Description

The "SubDivisionCount" property specifies the number of subdivisions of the main units.

Type

Int32

Access

Read-write

Syntax

```
XValueAxis.SubDivisionCount
```

See also

XValueAxis (Page 3032)

XValueAxis.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
XValueAxis.TickColor
```

See also

XValueAxis (Page 3032)

XValueAxis.Visible

Description

The "Visible" property specifies whether the value axis is visible.

Type

Bool

Access

Read-write

Syntax

```
XValueAxis.Visible
```

See also

XValueAxis (Page 3032)

FunctionTrendArea.FunctionTrends

Description

The "Trends" property returns the function trends ("FunctionTrend" objects) of the function trend area.

Type

Object, HmiFunctionTrendCollection (Page 3057)

Access

Read-only

Syntax

```
FunctionTrendArea.FunctionTrends
```

See also

FunctionTrendArea (Page 3028)

HmiFunctionTrendCollection (Page 3057)

HmiFunctionTrendCollection

Description

The "HmiFunctionTrendCollection" object is a list of all function trends ("FunctionTrend" objects) of the f(x) trend view.

Use

The "HmiFunctionTrendCollection" object is a list and can be counted and enumerated. You can access the "HmiFunctionTrendCollection" list using the index or the tag names.

Object type

HmiFunctionTrendCollection

Properties

The "HmiFunctionTrendCollection" object has the following properties:

- **Count**
  Returns the number of function trends of the "HmiFunctionTrendCollection" list.

Methods

The "HmiFunctionTrendCollection" object has the following methods:

- **Item()**
  Returns a function trend of the "HmiFunctionTrendCollection" list.

See also

FunctionTrendArea.FunctionTrends (Page 3057)

HmiFunctionTrendCollection.Count

Description

The "Count" property returns the number of function trends in the "HmiFunctionTrendCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiFunctionTrendCollection.Count
```

See also

HmiFunctionTrendCollection (Page 3057)

HmiFunctionTrendCollection.Item()

Description

The "Item" method returns a function trend of the "HmiFunctionTrendCollection" list.

Syntax

```
HmiFunctionTrendCollection[.Item](HmiFunctionTrendName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiFunctionTrendCollection" object.

Parameters

`HmiFunctionTrendName`

Type: String

Name of the function trend.

Return value

Object, HmiFunctionTrendPart (Page 3059)

See also

HmiFunctionTrendCollection (Page 3057)

FunctionTrend (Page 3059)

FunctionTrend

Description

The "FunctionTrend" object represents a trend of the trend area.

Object type

HmiFunctionTrendPart

Properties

The "FunctionTrend" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BeginTime**
  Specifies the date and time for the start time of the time range.
- **DashType**
  Specifies the stroke style of the function trend.
- **DataSourceX**
  Specifies the tag for the data source of the x value axis.
- **DataSourceY**
  Specifies the tag for the data source of the y value axis.
- **DisplayName**
  Specifies the display name of the function trend.
- **EndTime**
  Specifies the date and time for the end time of the time range.
- **LineColor**
  Specifies the function trend color.
- **LineWidth**
  Specifies the function trend width.
- **MarkerColor**
  Specifies the color of the function trend points.
- **MarkerDimension**
  Specifies the width of the function trend points.
- **MarkerGraphic**
  Specifies a graphic element as a function trend point.
- **MarkerType**
  Specifies the type of the function trend points.
- **PointCount**
  Specifies the number of measurement points from the start time.
- **QualityVisualization**
  Specifies the colors for values of a specific quality.
- **RangeType**
  Specifies the type of time range.
- **ShowLoggedDataImmediately**
  Specifies which logged values are displayed.
- **Thresholds**
  Returns the list of all limit values of the function trend.

- **TimeRangeBase**
  Specifies the base of the time range.
- **TimeRangeFactor**
  Specifies the factor of the time base for defining the time range.
- **TrendMode**
  Specifies the type of function trend display.
- **Visible**
  Specifies whether the function trend is visible.
- **XValueAxis**
  References an x value axis of the function trend.
- **YValueAxis**
  References a y value axis of the function trend.

Methods

--

See also

HmiFunctionTrendCollection (Page 3057)

HmiFunctionTrendCollection.Item() (Page 3059)

FunctionTrend.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrend.AlternateBackColor
```

See also

FunctionTrend (Page 3059)

FunctionTrend.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrend.BackColor
```

See also

FunctionTrend (Page 3059)

FunctionTrend.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient

- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
FunctionTrend.BackFillPattern
```

See also

FunctionTrend (Page 3059)

FunctionTrend.BeginTime

Description

The "BeginTime" property specifies the date and time for the start time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
FunctionTrend.BeginValue
```

See also

FunctionTrend (Page 3059)

FunctionTrend.DashType

Description

The "DashType" property specifies the stroke style of the function trend.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dot
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
FunctionTrend.DashType
```

See also

FunctionTrend (Page 3059)

FunctionTrend.DataSourceX

Description

The "DataSourceX" property specifies the tag for the data source of the x value axis.

Type

Object, HmiDataSourcePart

Access

Read-write

Syntax

```
FunctionTrend.DataSourceX
```

See also

FunctionTrend (Page 3059)

DataSource.Source

Description

The "Source" property specifies the data source, e.g. a tag or logging tag.

Type

String

Access

Read-write

Syntax

```
DataSource.Source
```

See also

FunctionTrend.DataSourceX (Page 3064)

DataSource.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
DataSource.VisualizeQuality
```

See also

FunctionTrend.DataSourceX (Page 3064)

FunctionTrend.DataSourceY

Description

The "DataSourceY" property specifies the tag for the data source of the y value axis.

Type

Object, HmiDataSourcePart

Access

Read-write

Syntax

```
FunctionTrend.DataSourceX
```

See also

FunctionTrend (Page 3059)

DataSource.Source

Description

The "Source" property specifies the data source, e.g. a tag or logging tag.

Type

String

Access

Read-write

Syntax

```
DataSource.Source
```

See also

FunctionTrend.DataSourceY (Page 3066)

DataSource.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
DataSource.VisualizeQuality
```

See also

FunctionTrend.DataSourceY (Page 3066)

FunctionTrend.DisplayName

Description

The "DisplayName" property specifies the display name of the function trend.

Type

String

Access

Read-write

Syntax

```
FunctionTrend.DisplayName
```

See also

FunctionTrend (Page 3059)

FunctionTrend.EndTime

Description

The "EndTime" property specifies the date and time for the end time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
FunctionTrend.EndValue
```

See also

FunctionTrend (Page 3059)

FunctionTrend.LineColor

Description

The "LineColor" property specifies the function trend color.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrend.LineColor
```

See also

FunctionTrend (Page 3059)

FunctionTrend.LineWidth

Description

The "LineWidth" property specifies the function trend width.

Type

UInt8

Access

Read-write

Syntax

```
FunctionTrend.LineWidth
```

See also

FunctionTrend (Page 3059)

FunctionTrend.MarkerColor

Description

The "MarkerColor" property specifies the color of the function trend points.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrend.MarkerColor
```

See also

FunctionTrend (Page 3059)

FunctionTrend.MarkerDimension

Description

The "MarkerDimension" property specifies the width of the function trend points.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrend.MarkerDimension
```

See also

FunctionTrend (Page 3059)

FunctionTrend.MarkerGraphic

Description

The "MarkerGraphic" property specifies a graphic element as a function trend point.

Type

String

Access

Read-write

Syntax

```
FunctionTrend.MarkerGraphic
```

See also

FunctionTrend (Page 3059)

FunctionTrend.MarkerType

Description

The "MarkerType" property specifies the type of function trend points.

Type

Int32, HmiMarkerType

Specifies the type of the trend points:

- None (0): None
- Points (1): Dots
- Square (2): Square
- Circle (3): Circles
- Graphic (4): Graphic

Access

Read-write

Syntax

```
FunctionTrend.MarkerType
```

See also

FunctionTrend (Page 3059)

FunctionTrend.PointCount

Description

The "PointCount" property specifies the number of measurement points from the start time.

Type

Int32

Access

Read-write

Syntax

```
FunctionTrend.PointCount
```

See also

FunctionTrend (Page 3059)

FunctionTrend.QualityVisualization

Description

The "QualityVisualization" property specifies the colors for values of a specific quality.

Type

Object, HmiQualityPart

Access

Read-write

Syntax

```
FunctionTrend.QualityVisualization
```

See also

FunctionTrend (Page 3059)

Quality.BadColor

Description

The "BadColor" property specifies the color for values of the quality "Bad". Values of this quality cannot be used.

Type

UInt32

Access

Read-write

Syntax

```
Quality.BadColor
```

See also

FunctionTrend.QualityVisualization (Page 3072)

Quality.UncertainColor

Description

The "UncertainColor" property specifies the color for values of the quality "Uncertain". The quality of this level's values is worse than usual. It might still be possible to use the values, however.

Type

UInt32

Access

Read-write

Syntax

```
Quality.UncertainColor
```

See also

FunctionTrend.QualityVisualization (Page 3072)

Quality.Visible

Description

The "Visible" property specifies whether the colors are visible.

Type

Bool

Access

Read-write

Syntax

```
Quality.Visible
```

See also

FunctionTrend.QualityVisualization (Page 3072)

FunctionTrend.RangeType

Description

The "RangeType" property specifies the type of time range.

Type

Int32, HmiTimeRangeType

Specifies the time range:

- TimeRange (0): Any time range
- FromBeginToEnd (1): Total time range
- PointCount (2): Number of measurement points

Access

Read-write

Syntax

```
FunctionTrend.RangeType
```

See also

FunctionTrend (Page 3059)

FunctionTrend.ShowLoggedDataImmediately

Description

The "ShowLoggedDataImmediately" property specifies which logged values are displayed:

- True: Entire visible range
- False: Only from current time

Type

Bool

Access

Read-write

Syntax

```
Trend.ShowLoggedDataImmediately
```

See also

FunctionTrend (Page 3059)

FunctionTrend.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the function trend.

Type

Object, HmiThresholdCollection (Page 3076)

Access

Read-only

Syntax

```
FunctionTrend.Thresholds
```

See also

FunctionTrend (Page 3059)

HmiThresholdCollection (Page 3076)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

FunctionTrend.Thresholds (Page 3075)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 3076)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 3078)

See also

HmiThresholdCollection (Page 3076)

Threshold (Page 3078)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.
- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

See also

HmiThresholdCollection (Page 3076)

HmiThresholdCollection.Item() (Page 3077)

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 3078)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 3078)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 3078)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold
- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 3078)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 3078)

FunctionTrend.TimeRangeBase

Description

The "TimeRangeBase" property specifies the base of the time range.

Type

Int32, HmiTimeRangeBase

Specifies a time range:

- Undefined (0): Not defined
- Millisecond (1): Millisecond
- Second (2): Second
- Minute (3): Minute
- Hour (4): Hour
- Day (5): Day
- Month (6): Month
- Year (7): Year

Access

Read-write

Syntax

```
FunctionTrend.TimeRangeBase
```

See also

FunctionTrend (Page 3059)

FunctionTrend.TimeRangeFactor

Description

The "TimeRangeFactor" property specifies the factor of the time base for defining the time range.

Type

Int32

Access

Read-write

Syntax

```
FunctionTrend.TimeRangeFactor
```

See also

FunctionTrend (Page 3059)

FunctionTrend.TrendMode

Description

The "TrendMode" property specifies the type of function trend display.

Type

Int32, HmiTrendMode

Specifies the trend type:

- Points (0): Dots
- Interpolated (1): Interpolated
- Stepped (2): Levels
- Bar (3): Bar
- Value (4): Values

Access

Read-write

Syntax

```
FunctionTrend.TrendMode
```

See also

FunctionTrend (Page 3059)

FunctionTrend.Visible

Description

The "Visible" property specifies whether the function trend is visible.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrend.Visible
```

See also

FunctionTrend (Page 3059)

FunctionTrend.XValueAxis

Description

The "XValueAxis" property references an x value axis of the function trend.

Type

Object, HmiXValueAxisPart (Page 3032)

Access

Read-write

Syntax

```
FunctionTrend.XValueAxis
```

See also

FunctionTrend (Page 3059)

XValueAxis (Page 3032)

XValueAxis

Description

XValueAxis (Page 3032)

FunctionTrend.YValueAxis

Description

The "YValueAxis" property references a y value axis of the function trend.

Type

Object, HmiYValueAxisPart (Page 3088)

Access

Read-write

Syntax

```
FunctionTrend.YValueAxis
```

See also

FunctionTrend (Page 3059)

YValueAxis (Page 3088)

YValueAxis

Description

YValueAxis (Page 3088)

FunctionTrendArea.GridLines

Description

The "GridLines" property specifies the grid lines of the function trend area.

Type

Int32, HmiGridLine

Specifies the display of the grid lines:

- None (0): None
- VerticalMajor (1): Vertical, coarse
- HorizontalMajor (2): Horizontal, coarse
- VerticalMinor (4): Vertical, fine
- HorizontalMinor (8): Horizontal, fine

Access

Read-write

Syntax

```
FunctionTrendArea.GridLines
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendArea.LeftValueAxes

Description

The "LeftValueAxes" property returns the left value axes of the function trend area.

Type

Object, HmiYValueAxisCollection (Page 3086)

Access

Read-only

Syntax

```
TrendArea.LeftValueAxes
```

See also

FunctionTrendArea (Page 3028)

HmiYValueAxisCollection (Page 3086)

HmiYValueAxisCollection

Description

The "HmiYValueAxisCollection" object is a list of all value axes ("YValueAxis" objects) of the trend area.

Use

The "HmiYValueAxisCollection" object is a list and can be counted and enumerated. You can access the "HmiYValueAxisCollection" list using the index or the tag name.

Object type

HmiYValueAxisCollection

Properties

The "HmiYValueAxisCollection" object has the following properties:

- **Count**
  Returns the number of value axes in the "HmiYValueAxisCollection" list.

Methods

The "HmiYValueAxisCollection" object has the following methods:

- **Item()**
  Returns a value axis of the "HmiYValueAxisCollection" list.

See also

TrendArea.LeftValueAxes (Page 6581)

HmiYValueAxisCollection (Page 3086)

HmiYValueAxisCollection.Count

Description

The "Count" property returns the number of value axes in the "HmiYValueAxisCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiYValueAxisCollection.Count
```

See also

HmiYValueAxisCollection (Page 3086)

HmiYValueAxisCollection.Item()

Description

The "Item" method returns a value axis of the "HmiYValueAxisCollection" list.

Syntax

```
HmiYValueAxisCollection[.Item](HmiYValueAxisName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiYValueAxisCollection" object.

Parameters

`HmiYValueAxisName`

Type: String

Name of the value axis

Return value

Object, HmiYValueAxisPart (Page 3088)

See also

HmiYValueAxisCollection (Page 3086)

YValueAxis (Page 3088)

YValueAxis

Description

The "YValueAxis" object represents a value axis of the trend area.

Object type

HmiYValueAxisPart

Properties

The "YValueAxis" object has the following properties:

- **ApplyScalingEntries**
  Specifies whether the user scaling of the axis section is applied.
- **AutoRange**
  Specifies whether automatic determination of the value range is activated by the minimum and maximum value of the trend.
- **AutoScaling**
  Specifies whether the automatic scaling is activated.
- **AxisColor**
  Specifies the color of the value axis.
- **BeginValue**
  Specifies the start of a value range or value range section.
- **DisplayName**
  Specifies the display name of the value axis.
- **DivisionCount**
  Specifies the number of main units with subdivisions.
- **EndValue**
  Specifies the end of a value range or value range section.
- **HelpLines**
  Returns the appearance of the help lines.
- **LabelColor**
  Specifies the color of the axis labeling.

- **LabelFont**
  Specifies the font of the axis labeling.
- **LargeTickLabelingStep**
  Specifies the interval at which scale sections are labeled.
- **MeasurementUnit**
  Returns the displayed unit.
- **MeasurementUnitType**
  Specifies the display format of the unit.
- **OutputFormat**
  Specifies the format for displaying the axis values.
- **ScalingEntries**
  Returns the specification of the user scaling of the axis sections.
- **ScaleMode**
  Specifies the scale mode:
- **ScalingType**
  Specifies the scaling.
- **ShowScalingDisplayNames**
  Specifies whether the display names of the user scaling are used.
- **SubDivisionCount**
  Specifies the number of divisions of the main units.
- **TickColor**
  Specifies the color of the tick marks.
- **Visible**
  Specifies whether the value axis is visible.

Methods

--

See also

HmiYValueAxisCollection (Page 3086)

HmiYValueAxisCollection.Item() (Page 3087)

YValueAxis.ApplyScalingEntries

Description

The "ApplyScalingEntries" property specifies whether the user scaling of the axis sections is applied.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.ApplyScalingEntries
```

See also

YValueAxis (Page 3088)

YValueAxis.AutoRange

Description

The "AutoRange" property specifies whether automatic determination of the value range is activated by the minimum and maximum value of the trend.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.AutoRange
```

See also

YValueAxis (Page 3088)

YValueAxis.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.AutoScaling
```

See also

YValueAxis (Page 3088)

YValueAxis.AxisColor

Description

The "AxisColor" property specifies the color of the value axis.

Type

UInt32

Access

Read-write

Syntax

```
YValueAxis.BackColor
```

See also

YValueAxis (Page 3088)

YValueAxis.BeginValue

Description

The "BeginValue" property specifies the start of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
YValueAxis.BeginValue
```

See also

YValueAxis (Page 3088)

YValueAxis.DisplayName

Description

The "DisplayName" property specifies the display name of the value axis.

Type

String

Access

Read-write

Syntax

```
YValueAxis.DisplayName
```

See also

YValueAxis (Page 3088)

YValueAxis.DivisionCount

Description

The "DivisionCount" property specifies the number of main units with subdivisions. To this purpose the automatic scaling must be switched off.

Type

Int32

Access

Read-write

Syntax

```
YValueAxis.DivisionCount
```

See also

YValueAxis (Page 3088)

YValueAxis.EndValue

Description

The "EndValue" property specifies the end of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
YValueAxis.EndValue
```

See also

YValueAxis (Page 3088)

YValueAxis.HelpLines

Description

The "HelpLines" property returns the appearance of the help lines.

Type

Object, HmiHelpLineCollection (Page 3094)

Access

Read-only

Syntax

```
YValueAxis.HelpLines
```

See also

YValueAxis (Page 3088)

HmiHelpLineCollection (Page 3094)

HmiHelpLineCollection

Description

The "HmiHelpLineCollection" object is a list of all help lines ("HelpLine" objects).

Use

The "HmiHelpLineCollection" object is a list and can be counted and enumerated. You can access the "HmiHelpLineCollection" list using the index or the tag name.

Object type

HmiHelpLineCollection

Properties

The "HmiHelpLineCollection" object has the following properties:

- **Count**
  Returns the number of help lines of the "HmiHelpLineCollection" list.

Methods

The "HmiHelpLineCollection" object has the following methods:

- **Item()**
  Returns a help line of the "HmiHelpLineCollection" list.

See also

YValueAxis.HelpLines (Page 3093)

HmiHelpLineCollection.Count

Description

The "Count" property returns the number of help lines in the "HmiHelpLineCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiHelpLineCollection.Count
```

See also

HmiHelpLineCollection (Page 3094)

HmiHelpLineCollection.Item()

Description

The "Item" method returns help line of the "HmiHelpLineCollection" list.

Syntax

```
HmiHelpLineCollection[.Item](HmiHelpLineName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiHelpLineCollection" object.

Parameters

`HmiHelpLineName`

Type: String

Name of the help line

Return value

Object, HmiHelpLinePart (Page 3096)

See also

HmiHelpLineCollection (Page 3094)

HelpLine (Page 3096)

HelpLine

Description

The "HelpLine" object represents a help line of the value axis.

Object type

HmiHelpLinePart

Properties

The "HelpLine" object has the following properties:

- **Value**
  Specifies the value of the help line.
- **Visible**
  Specifies whether the help line is visible.

Methods

--

See also

HmiHelpLineCollection (Page 3094)

HmiHelpLineCollection.Item() (Page 3095)

HelpLine.Value

Description

The "Value" property sets the value of the help line.

Type

Float

Access

Read-write

Syntax

```
HelpLine.Value
```

See also

HelpLine (Page 3096)

HelpLine.Visible

Description

The "Visible" property specifies whether the help line is visible.

Type

Bool

Access

Read-write

Syntax

```
HelpLine.Visible
```

See also

HelpLine (Page 3096)

YValueAxis.LabelColor

Description

The "LabelColor" property specifies the color of the axis labeling.

Type

UInt32

Access

Read-write

Syntax

```
YValueAxis.LabelColor
```

See also

YValueAxis (Page 3088)

YValueAxis.LabelFont

Description

The "LabelFont" property specifies the font of the axis labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
YValueAxis.LabelFont
```

See also

YValueAxis (Page 3088)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

YValueAxis.LabelFont (Page 3098)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

YValueAxis.LabelFont (Page 3098)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

YValueAxis.LabelFont (Page 3098)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

YValueAxis.LabelFont (Page 3098)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

YValueAxis.LabelFont (Page 3098)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

YValueAxis.LabelFont (Page 3098)

YValueAxis.LargeTickLabelingStep

Description

The "LargeTickLabelingStep" property specifies the interval at which scale sections are labeled.

Type

UInt8

Access

Read-write

Syntax

```
YValueAxis.LargeTickLabelingStep
```

See also

YValueAxis (Page 3088)

YValueAxis.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
YValueAxis.MeasurementUnit
```

See also

YValueAxis (Page 3088)

YValueAxis.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
YValueAxis.MeasurementUnitType
```

See also

YValueAxis (Page 3088)

YValueAxis.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the axis values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
YValueAxis.OutputFormat
```

See also

YValueAxis (Page 3088)

YValueAxis.ScalingEntries

Description

The "ScalingEntries" property returns the user scaling of the axes sections.

Type

Object, HmiScalingEntryCollection (Page 3104)

Access

Read-only

Syntax

```
YValueAxis.ScalingEntries
```

See also

YValueAxis (Page 3088)

HmiScalingEntryCollection (Page 3104)

HmiScalingEntryCollection

Description

The "HmiScalingEntryCollection" object is a list of all user-defined axis sections ("ScalingEntry" objects).

Use

The "HmiScalingEntryCollection" object is a list and can be counted and enumerated. You can access the "HmiScalingEntryCollection" list using the index or the tag name.

Object type

HmiHelpLineCollection

Properties

The "HmiScalingEntryCollection" object has the following properties:

- **Count**
  Returns the number of user-defined axis sections of the "HmiScalingEntryCollection" list.

Methods

The "HmiScalingEntryCollection" object has the following methods:

- **Item()**
  Returns a user-defined axis section of the "HmiScalingEntryCollection" list.

See also

YValueAxis.ScalingEntries (Page 3103)

HmiScalingEntryCollection.Count

Description

The "Count" property returns the number of the user-defined axis sections in the "HmiScalingEntryCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiScalingEntryCollection.Count
```

See also

HmiScalingEntryCollection (Page 3104)

HmiScalingEntryCollection.Item()

Description

The "Item" method returns a user-defined axis section of the "HmiScalingEntryCollection" list.

Syntax

```
HmiScalingEntryCollection[.Item](HmiScalingEntryName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiScalingEntryCollection" object.

Parameters

`HmiScalingEntryName`

Type: String

Name of the user-defined axis section

Return value

Object, HmiScalingEntryPart (Page 3106)

See also

HmiScalingEntryCollection (Page 3104)

Scaling Entry (Page 3106)

Scaling Entry

Description

The "ScalingEntry" object represents a user-defined axis section of the value axis.

Object type

HmiScalingEntryPart

Properties

The "YValueAxis" object has the following properties:

- **BeginValue**
  Specifies the start of a value range section.
- **BeginValueTarget**
  Specifies the scaled value for the start of a value range section.
- **DisplayName**
  Specifies the display name of an axis section.
- **EndValue**
  Specifies the end of a value range section.
- **EndValueTarget**
  Specifies the scaled value for the end of a value range section.

Methods

--

See also

HmiScalingEntryCollection (Page 3104)

HmiScalingEntryCollection.Item() (Page 3105)

ScalingEntry.BeginValue

Description

The "BeginValue" property specifies the start of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.BeginValue
```

See also

Scaling Entry (Page 3106)

ScalingEntry.BeginValueTarget

Description

The"BeginValueTarget" property specifies the scaled value for the start of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.BeginValueTarget
```

See also

Scaling Entry (Page 3106)

ScalingEntry.DisplayName

Description

The "DisplayName" property specifies the display name of an axis section.

Type

String

Access

Read-write

Syntax

```
ScalingEntry.DisplayName
```

See also

Scaling Entry (Page 3106)

ScalingEntry.EndValue

Description

The "EndValue" property specifies the end of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.EndValue
```

See also

Scaling Entry (Page 3106)

ScalingEntry.EndValueTarget

Description

The"EndValueTarget" property specifies the scaled value for the end of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.EndValueTarget
```

See also

Scaling Entry (Page 3106)

YValueAxis.ScaleMode

Description

The "ScaleMode" property specifies the scale mode.

Type

Int32, HmiScaleMode

Specifies the scale mode:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
YValueAxis.ScaleMode
```

See also

YValueAxis (Page 3088)

YValueAxis.ScalingType

Description

The "ScalingType" property specifies the scaling.

Type

Int32, HmiScalingType

Specifies the scaling:

- Linear (0): Linear
- Logarithmic (1): Logarithmic
- NegativeLogarithmic (2): Negative logarithmic
- Tangent (4): Tangential
- Quadratic (5): Square
- Cubic (6): Cubic

Access

Read-write

Syntax

```
YValueAxis.ScalingType
```

See also

YValueAxis (Page 3088)

YValueAxis.ShowScalingDisplayNames

Description

The "ShowScalingDisplayNames" property specifies whether the display names of the user scaling are used.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.ShowScalingDisplayNames
```

See also

YValueAxis (Page 3088)

YValueAxis.SubDivisionCount

Description

The "SubDivisionCount" property specifies the number of subdivisions of the main units.

Type

Int32

Access

Read-write

Syntax

```
YValueAxis.SubDivisionCount
```

See also

YValueAxis (Page 3088)

YValueAxis.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
YValueAxis.TickColor
```

See also

YValueAxis (Page 3088)

YValueAxis.Visible

Description

The "Visible" property specifies whether the value axis is visible.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.Visible
```

See also

YValueAxis (Page 3088)

FunctionTrendArea.MajorGridLinesColor

Description

The "MajorGridLinesColor" property specifies the color of the main grid lines.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendArea.MajorGridLinesColor
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendArea.MinorGridLinesColor

Description

The "MinorGridLinesColor" specifies the color of the auxiliary grid lines.

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendArea.MinorGridLinesColor
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendArea.Name

Description

The "Name" property returns the name of the function trend area.

Type

String

Access

Read-only

Syntax

```
FunctionTrendArea.Name
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendArea.RightValueAxes

Description

The "RightValueAxes" property returns the right value axes of the function trend area.

Type

Object, HmiYValueAxisCollection (Page 3086)

Access

Read-only

Syntax

```
FunctionTrendArea.RightValueAxes
```

See also

FunctionTrendArea (Page 3028)
HmiYValueAxisCollection (Page 3086)

YValueAxisCollection

Description

YValueAxisCollection (Page 3086)

FunctionTrendArea.Ruler

Description

The "Ruler" property specifies the appearance of the ruler to determine the function trend value.

Type

Object, HmiRulerPart

Access

Read-write

Syntax

```
FunctionTrendArea.Ruler
```

See also

FunctionTrendArea (Page 3028)

Ruler.Color

Description

The "Color" property specifies the color of the ruler.

Type

UInt32

Access

Read-write

Syntax

```
Ruler.Color
```

See also

FunctionTrendArea.Ruler (Page 3114)

Ruler.Width

Description

The "Width" property specifies the width of the ruler.

Type

UInt32

Access

Read-write

Syntax

```
Ruler.Width
```

See also

FunctionTrendArea.Ruler (Page 3114)

FunctionTrendArea.SelectedFunctionTrend

Description

The "SelectedFunctionTrend" property specifies the selected function trend of the function trend area.

Type

Object, HmiFunctionTrendPart (Page 3059)

Access

Read-write

Syntax

```
FunctionTrendArea.SelectedFunctionTrend
```

See also

FunctionTrendArea (Page 3028)

FunctionTrend (Page 3059)

FunctionTrend

Description

FunctionTrend (Page 3059)

FunctionTrendArea.SizeFactor

Description

The "SizeFactor" property specifies the scaling factor of the function trend area relative to its height.

Type

UInt16

Access

Read-write

Syntax

```
FunctionTrendArea.SizeFactor
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendArea.TopValueAxes

Description

The "TopValueAxes" property returns the upper value axes of the function trend area.

Type

Object, HmiXValueAxisCollection (Page 3030)

Access

Read-only

Syntax

```
FunctionTrendArea.TopValueAxes
```

See also

FunctionTrendArea (Page 3028)

HmiXValueAxisCollection (Page 3030)

XValueAxisCollection

Description

XValueAxisCollection (Page 3030)

FunctionTrendArea.Visible

Description

The "Visible" property specifies whether the function trend area is visible.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendArea.Visible
```

See also

FunctionTrendArea (Page 3028)

FunctionTrendControl.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendControl.Height
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Icon

Description

The "Icon" property specifies the icon of the f(x) trend view.

Type

String

Access

Read-write

Syntax

```
FunctionTrendControl.Icon
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Layer

Description

The "Layer" property returns the layer of the screen in which the f(x) trend view is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
FunctionTrendControl.Layer
```

See also

FunctionTrendControl (Page 3014)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

FunctionTrendControl.Layer (Page 3119)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

FunctionTrendControl.Layer (Page 3119)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

FunctionTrendControl.Layer (Page 3119)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

FunctionTrendControl.Layer (Page 3119)

FunctionTrendControl.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
FunctionTrendControl.Left
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Legend

Description

The "Legend" property specifies the appearance of the legend ("Legend" object).

Type

Object, HmiLegendPart

Access

Read-write

Syntax

```
FunctionTrendControl.Legend
```

See also

FunctionTrendControl (Page 3014)

Legend.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Legend.ForeColor
```

See also

FunctionTrendControl.Legend (Page 3122)

Legend.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Legend.Font
```

See also

FunctionTrendControl.Legend (Page 3122)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Legend.Font (Page 3123)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Legend.Font (Page 3123)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Legend.Font (Page 3123)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Legend.Font (Page 3123)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Legend.Font (Page 3123)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Legend.Font (Page 3123)

Legend.Visible

Description

The "Visible" property specifies whether the legend is visible.

Type

Bool

Access

Read-write

Syntax

```
Legend.Visible
```

See also

FunctionTrendControl.Legend (Page 3122)

FunctionTrendControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
FunctionTrendControl.Margin
```

See also

FunctionTrendControl (Page 3014)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

FunctionTrendControl.Margin (Page 3127)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

FunctionTrendControl.Margin (Page 3127)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

FunctionTrendControl.Margin (Page 3127)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

FunctionTrendControl.Margin (Page 3127)

FunctionTrendControl.Name

Description

The "Name" property returns the name of the f(x) trend view.

Type

String

Access

Read-only

Syntax

```
FunctionTrendControl.Name
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Online

Description

The "Online" property specifies the start and stop of the update of the f(x) trend view.

- True: Online. The f(x) trend view is updated with new values.
- False: Offline. No new values are added to the f(x) trend view.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.Online
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
FunctionTrendControl.Parent
```

See also

FunctionTrendControl (Page 3014)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

FunctionTrendControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the f(x) trend view was created.

Type

String

Access

Read-only

Syntax

```
FunctionTrendControl.RenderingTemplate
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.ShiftAxis

Description

The "ShiftAxis" property specifies whether to swap the x axis and y axis of the f(x) trend view.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.ShiftAxis
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the f(x) trend view is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.ShowFocusVisual
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.ShowRuler

Description

The "ShowRuler" property specifies whether the ruler is displayed for determining a function trend value.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.ShowRuler
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the f(x) trend view.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
FunctionTrendControl.StatusBar
```

See also

FunctionTrendControl (Page 3014)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

FunctionTrendControl.StatusBar (Page 3134)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 3135)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

HmiControlBarElementCollection (Page 3135)

FunctionTrendControl.StatusBar (Page 3134)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 3135)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 3135)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6421)

See also

HmiControlBarElementCollection (Page 3135)

Control Bar Elements (Page 6421)

Control Bar Elements

Description

Control Bar Elements (Page 3152)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

FunctionTrendControl.StatusBar (Page 3134)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

FunctionTrendControl.StatusBar (Page 3134)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 3138)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 3138)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 3138)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 3138)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 3138)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 3138)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

FunctionTrendControl.StatusBar (Page 3134)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 3142)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 3142)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 3142)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 3142)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

FunctionTrendControl.StatusBar (Page 3134)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 3144)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 3144)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 3144)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 3144)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

FunctionTrendControl.StatusBar (Page 3134)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

FunctionTrendControl.StatusBar (Page 3134)

FunctionTrendControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the f(x) trend view.

Type

String

Access

Read-only

Syntax

```
FunctionTrendControl.StyleItemClass
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.TabIndex

Description

The "TabIndex" property returns the position of the f(x) trend view in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
FunctionTrendControl.TabIndex
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the f(x) trend view.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
FunctionTrendControl.ToolBar
```

See also

FunctionTrendControl (Page 3014)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

FunctionTrendControl.ToolBar (Page 3149)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 3150)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

HmiControlBarElementCollection (Page 3150)

FunctionTrendControl.ToolBar (Page 3149)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 3150)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 3152)

See also

HmiControlBarElementCollection (Page 3150)

Control Bar Elements (Page 3152)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item() (Page 3151)

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 3152)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 3152)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 3152)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 3152)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 3152)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 3158)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 3158)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 3158)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 3158)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 3158)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 3158)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 3158)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 3158)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 3152)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 3152)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 3152)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 3168)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 3168)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 3168)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 3168)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 3152)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 3152)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 3152)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 3152)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 3173)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 3173)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 3173)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 3173)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 3152)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 3152)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 3152)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item() (Page 3151)

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 3178)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 3178)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 3180)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 3180)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered

- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 3180)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 3180)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 3180)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 3180)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 3180)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 3180)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control

- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms

- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 3178)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 3178)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 3190)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 3190)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 3190)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 3190)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 3178)

ControlBarButton.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 3178)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 3196)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 3196)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 3196)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 3196)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 3178)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 3178)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.

- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item() (Page 3151)

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display

- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status

- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 3201)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 3207)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 3207)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 3207)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 3207)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 3201)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 3213)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 3213)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 3213)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 3213)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 3201)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 3201)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.

- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item() (Page 3151)

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 3218)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 3224)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 3224)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 3224)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 3224)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 3218)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 3229)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 3229)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 3229)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 3229)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 3218)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 3218)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item() (Page 3151)

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 3234)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 3243)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 3243)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 3243)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 3243)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 3234)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 3248)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 3248)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 3248)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 3248)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 3234)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 3234)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 3150)

HmiControlBarElementCollection.Item() (Page 3151)

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 3254)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 3261)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 3261)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 3254)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 3272)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 3272)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 3272)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 3272)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 3254)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 3277)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 3277)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 3277)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 3277)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 3254)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 3254)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

FunctionTrendControl.ToolBar (Page 3149)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

FunctionTrendControl.ToolBar (Page 3149)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 3283)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 3283)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 3283)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 3283)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 3283)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 3283)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

FunctionTrendControl.ToolBar (Page 3149)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 3286)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 3286)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 3286)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 3286)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

FunctionTrendControl.ToolBar (Page 3149)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 3289)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 3289)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 3289)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 3289)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

FunctionTrendControl.ToolBar (Page 3149)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

FunctionTrendControl.ToolBar (Page 3149)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

FunctionTrendControl.ToolBar (Page 3149)

FunctionTrendControl.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
FunctionTrendControl.Top
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Visible

Description

The "Visible" property specifies whether the f(x) trend view is visible.

Type

Bool

Access

Read-write

Syntax

```
FunctionTrendControl.Visible
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
FunctionTrendControl.Width
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the f(x) trend view.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title

- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
FunctionTrendControl.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the f(x) trend view.

Syntax

```
FunctionTrendControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the f(x) trend view.

Syntax

```
FunctionTrendControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
FunctionTrendControl.PropertyFlashing(propertyName, enable[, value]
[, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl_OnActivated()

Description

The "OnActivated" event occurs when a f(x) trend view receives focus:

- An f(x) trend view is selected via the configured tab sequence.
- An f(x) trend view that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
FunctionTrendControl_OnActivated(item)
```

Context

`item`

Type: Object

f(x) trend view where the event occurs.

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the f(x) trend view.

Syntax

```
FunctionTrendControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

f(x) trend view where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when an f(x) trend view loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

Syntax

```
FunctionTrendControl_OnDeactivated(item)
```

Context

item

Type: Object

f(x) trend view where the event occurs.

See also

FunctionTrendControl (Page 3014)

FunctionTrendControl_OnInitialized()

Description

The "OnInitialized" event occurs when an f(x) trend view has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
FunctionTrendControl_OnInitialized(item)
```

Context

item

Type: Object

f(x) trend view where the event occurs.

See also

FunctionTrendControl (Page 3014)

Gauge

Description

The "Gauge" object represents a pointer instrument as an analog dial gauge for monitoring process values in runtime.

Object type

HmiGauge

Properties

The "Gauge" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **ComputedMaxPeakValue**
  Returns the highest process value that occurred.
- **ComputedMinPeakValue**
  Returns the lowest process value that occurred.
- **ComputedValueTendency**
  Returns the change of the process value.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the gauge.
- **CurvedScale**
  Specifies the scale of the gauge.
- **Enabled**
  Specifies whether the gauge can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **Height**
  Specifies the height.
- **Label**
  Specifies the labeling of the gauge.
- **Layer**
  Returns the layer of the screen in which the gauge is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.

- **Name**
  Returns the name of the gauge.
- **NormalRangeColor**
  Specifies the color of the normal range.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the gauge is operable.
- **OriginValue**
  Specifies the output value of the normal range that is visualized.
- **OutputFormat**
  Specifies the format for displaying the process values.
- **Parent**
  Returns the higher-level screen object.
- **PeakIndicators**
  Specifies whether the highest and lowest process value up to this time are displayed.
- **ProcessValue**
  Specifies the process value.
- **ProcessValueIndicatorBackColor**
  Specifies the background color for the process value.
- **ProcessValueIndicatorForeColor**
  Specifies the foreground color for the process value.
- **ProcessValueIndicatorMode**
  Specifies the type of display of the current process value.
- **RelativeToOrigin**
  Specifies whether the output value is an absolute or a percentage value between the minimum and maximum value.
- **RenderingTemplate**
  Returns the name of the template from which the gauge was created.
- **RequireExplicitUnlock**
  Returns whether the gauge is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the gauge rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ScaleBackColor**
  Specifies the background color of the scale.

- **ScaleForeColor**
  Specifies the foreground color of the scale.
- **ShowFocusVisual**
  Specifies whether the gauge is highlighted when in focus.
- **ShowTrendIndicator**
  Specifies whether the tendency (rising or falling) of the process value to be monitored is indicated by means of a small arrow.
- **StyleItemClass**
  Returns the style which is applied to the gauge.
- **TabIndex**
  Returns the position of the gauge in the tab sequence.
- **ThresholdIndicators**
  Specifies how parameterized limit values are visualized.
- **Thresholds**
  Returns the list of all limit values of the gauge.
- **Title**
  Specifies the caption which appears as the title.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **TrendIndicatorColor**
  Specifies the color of the trend indicator.
- **Visible**
  Specifies whether the gauge is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "Gauge" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the gauge.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Gauge" object has the following events:

- **OnActivated()**
  Occurs when a gauge receives focus.
- **OnContextTapped()**
  Occurs when a gauge is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a gauge loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the gauge is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the gauge is in focus.
- **OnTapped()**
  Occurs when a gauge is left-clicked or short-touched.

Gauge.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.AlternateBackColor
```

See also

Gauge (Page 3300)

Gauge.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Gauge.AlternateBorderColor
```

See also

Gauge (Page 3300)

Gauge.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Gauge.Authorization
```

See also

Gauge (Page 3300)

Gauge.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.BackColor
```

See also

Gauge (Page 3300)

Gauge.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.BorderColor
```

See also

Gauge (Page 3300)

Gauge.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Gauge.BorderWidth
```

See also

Gauge (Page 3300)

Gauge.ComputedMaxPeakValue

Description

The "ComputedMaxPeakValue" property returns the highest process value that occurred.

Type

Variant

Access

Read-only

Syntax

```
Gauge.ComputedMaxPeakValue
```

See also

Gauge (Page 3300)

Gauge.ComputedMinPeakValue (Page 3307)

Gauge.ComputedMinPeakValue

Description

The "ComputedMinPeakValue" property returns the lowest process value which occurred.

Type

Variant

Access

Read-only

Syntax

```
Gauge.ComputedMinPeakValue
```

See also

Gauge (Page 3300)

Gauge.ComputedMaxPeakValue (Page 3307)

Gauge.ComputedValueTendency

Description

The "ComputedValueTendency" property returns the change in the process value.

Type

Int32, HmiValueTendency

Returns the modification:

- Steady (0): No change
- Upwards (1): Change upwards
- Downwards (2): Change downwards

Access

Read-only

Syntax

```
Gauge.ComputedValueTendency
```

See also

Gauge (Page 3300)

Gauge.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the gauge.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Gauge.CurrentQuality
```

See also

Gauge (Page 3300)

Gauge.CurvedScale

Description

The "CurvedScale" property specifies the scale of the gauge.

Type

Object, HmiCurvedScalePart

Access

Read-write

Syntax

```
Gauge.CurvedScale
```

See also

Gauge (Page 3300)

CurvedScale.AngleRange

Description

The "AngleRange" property specifies the arc angle of the displayed scale clockwise.

Type

Int32

Access

Read-write

Syntax

```
CurvedScale.AngleRange
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
CurvedScale.AutoScaling
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.BeginValue

Description

The "BeginValue" property specifies the start of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
CurvedScale.BeginValue
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.DivisionCount

Description

The "DivisionCount" property specifies the number of main units with subdivisions. To this purpose the automatic scaling must be switched off.

Type

Int32

Access

Read-write

Syntax

```
CurvedScale.DivisionCount
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.EndValue

Description

The "EndValue" property specifies the end of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
CurvedScale.EndValue
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.LabelColor

Description

The "LabelColor" property specifies the color of the labeling.

Type

UInt32

Access

Read-write

Syntax

```
CurvedScale.LabelColor
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.LabelFont

Description

The "LabelFont" property specifies the font of the labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
CurvedScale.LabelFont
```

See also

Gauge.CurvedScale (Page 3309)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

CurvedScale.LabelFont (Page 3313)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

CurvedScale.LabelFont (Page 3313)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

CurvedScale.LabelFont (Page 3313)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

CurvedScale.LabelFont (Page 3313)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

CurvedScale.LabelFont (Page 3313)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

CurvedScale.LabelFont (Page 3313)

CurvedScale.LargeTickLabelingStep

Description

The "LargeTickLabelingStep" property specifies the interval at which scale sections are labeled.

Type

UInt8

Access

Read-write

Syntax

```
CurvedScale.LargeTickLabelingStep
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
CurvedScale.MeasurementUnit
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
CurvedScale.MeasurementUnitType
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
CurvedScale.OutputFormat
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.ScaleMode

Description

The "ScaleMode" property specifies the type of scaling.

Type

Int32, HmiScaleMode

Specifies the scaling:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
CurvedScale.ScaleMode
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.ScalingType

Description

The "ScalingType" property specifies the scaling.

Type

Int32, HmiScalingType

Specifies the scaling:

- Linear (0): Linear
- Logarithmic (1): Logarithmic
- NegativeLogarithmic (2): Negative logarithmic
- Tangent (4): Tangential
- Quadratic (5): Square
- Cubic (6): Cubic

Access

Read-write

Syntax

```
CurvedScale.ScalingType
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.StartAngle

Description

The "StartAngle" specifies the angle by which the start point deviates from the zero position (0° corresponds to 3 o'clock).

Type

Int32

Access

Read-write

Syntax

```
CurvedScale.StartAngle
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.SubDivisionCount

Description

The "SubDivisionCount" property specifies the number of subdivisions of the main units.

Type

Int32

Access

Read-write

Syntax

```
CurvedScale.SubDivisionCount
```

See also

Gauge.CurvedScale (Page 3309)

CurvedScale.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
CurvedScale.TickColor
```

See also

Gauge.CurvedScale (Page 3309)

Gauge.Enabled

Description

The "Enabled" property specifies whether the gauge can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Gauge.Enabled
```

See also

Gauge (Page 3300)

Gauge.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Gauge.Font
```

See also

Gauge (Page 3300)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Gauge.Font (Page 3322)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Gauge.Font (Page 3322)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Gauge.Font (Page 3322)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Gauge.Font (Page 3322)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Gauge.Font (Page 3322)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Gauge.Font (Page 3322)

Gauge.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Gauge.Height
```

See also

Gauge (Page 3300)

Gauge.Label

Description

The "Label" property specifies the labeling of the gauge.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Gauge.Label
```

See also

Gauge (Page 3300)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Gauge.Label (Page 3326)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 3326)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 3326)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 3326)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 3326)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 3326)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 3326)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Gauge.Label (Page 3326)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Gauge.Label (Page 3326)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Gauge.Label (Page 3326)

Gauge.Layer

Description

The "Layer" property returns the layer of the screen in which the gauge is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Gauge.Layer
```

See also

Gauge (Page 3300)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Gauge.Layer (Page 3332)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Gauge.Layer (Page 3332)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Gauge.Layer (Page 3332)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Gauge.Layer (Page 3332)

Gauge.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Gauge.Left
```

See also

Gauge (Page 3300)

Gauge.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Gauge.Margin
```

See also

Gauge (Page 3300)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Gauge.Margin (Page 3335)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Gauge.Margin (Page 3335)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Gauge.Margin (Page 3335)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Gauge.Margin (Page 3335)

Gauge.Name

Description

The "Name" property returns the name of the gauge.

Type

String

Access

Read-only

Syntax

```
Gauge.Name
```

See also

Gauge (Page 3300)

Gauge.NormalRangeColor

Description

The "NormalRangeColor" property specifies the color of the normal range.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.NormalRangeColor
```

See also

Gauge (Page 3300)

Gauge.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Gauge.Opacity
```

See also

Gauge (Page 3300)

Gauge.Operability

Description

The "Operability" property returns whether the gauge is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Gauge.Operability
```

See also

Gauge (Page 3300)

Gauge.OriginValue

Description

The "OriginValue" property specifies the output value of the normal range to be visualized.

Type

Float

Access

Read-write

Syntax

```
Gauge.OriginValue
```

See also

Gauge (Page 3300)

Gauge.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
Gauge.OutputFormat
```

See also

Gauge (Page 3300)

Gauge.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Gauge.Parent
```

See also

Gauge (Page 3300)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Gauge.PeakIndicators

Description

The "PeakIndicators" property specifies whether the highest and lowest process value up to this time are displayed.

Type

Int32, HmiPeakIndicator

Specifies the display of the peak indicator:

- None (0): No display
- Low (1): Only the lowest process value
- High (2): Only the highest process value

Access

Read-write

Syntax

```
Gauge.PeakIndicators
```

See also

Gauge (Page 3300)

Gauge.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
Gauge.ProcessValue
```

See also

Gauge (Page 3300)

Gauge.ProcessValueIndicatorBackColor

Description

The "ProcessValueIndicatorBackColor" property specifies the background color for the process value indicator.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.ProcessValueIndicatorBackColor
```

See also

Gauge (Page 3300)

Gauge.ProcessValueIndicatorForeColor (Page 3342)

Gauge.ProcessValueIndicatorForeColor

Description

The "ProcessValueIndicatorForeColor" property specifies the foreground color for the process value indicator.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.ProcessValueIndicatorForeColor
```

See also

Gauge (Page 3300)

Gauge.ProcessValueIndicatorBackColor (Page 3342)

Gauge.ProcessValueIndicatorMode

Description

The "ProcessValueIndicatorMode" property specifies the type of display of the current process value.

Type

Int32, HmiProcessIndicatorMode

Specifies the type of display:

- Bar (0): Bar only
- Indicator (1): Hair line or needle, no numerical display of the process value.
- DetailedIndicator (2): Detailed display with numerical value
- BarWithDetailedIndicator (3): Bar with numerical value

Access

Read-write

Syntax

```
Gauge.ProcessValueIndicatorMode
```

See also

Gauge (Page 3300)

Gauge.RelativeToOrigin

Description

The "RelativeToOrigin" property specifies whether the output value is an absolute or a percentage value between minimum and maximum value.

Type

Bool

Access

Read-write

Syntax

```
Gauge.RelativeToOrigin
```

See also

Gauge (Page 3300)

Gauge.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the gauge was created.

Type

String

Access

Read-only

Syntax

```
Gauge.RenderingTemplate
```

See also

Gauge (Page 3300)

Gauge.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the gauge can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Gauge.RequireExplicitUnlock
```

See also

Gauge (Page 3300)

Gauge.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Gauge.RotationAngle
```

See also

Gauge (Page 3300)

Gauge.RotationCenterPlacement (Page 3346)

Gauge.RotationCenterX (Page 3346)

Gauge.RotationCenterY (Page 3347)

Gauge.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the gauge rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Gauge.RotationCenterPlacement
```

See also

Gauge (Page 3300)

Gauge.RotationAngle (Page 3345)

Gauge.RotationCenterX (Page 3346)

Gauge.RotationCenterY (Page 3347)

Gauge.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Gauge.RotationCenterX
```

See also

Gauge (Page 3300)

Gauge.RotationAngle (Page 3345)

Gauge.RotationCenterPlacement (Page 3346)

Gauge.RotationCenterY (Page 3347)

Gauge.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Gauge.RotationCenterY
```

See also

Gauge (Page 3300)

Gauge.RotationAngle (Page 3345)

Gauge.RotationCenterPlacement (Page 3346)

Gauge.RotationCenterX (Page 3346)

Gauge.ScaleBackColor

Description

The "ScaleBackColor" property specifies the background color of the scale.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.ScaleBackColor
```

See also

Gauge (Page 3300)

Gauge.ScaleForeColor (Page 3348)

Gauge.ScaleForeColor

Description

The "ScaleForeColor" property specifies the foreground color of the scale.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.ScaleForeColor
```

See also

Gauge (Page 3300)

Gauge.ScaleBackColor (Page 3348)

Gauge.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the gauge is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Gauge.ShowFocusVisual
```

See also

Gauge (Page 3300)

Gauge.ShowTrendIndicator

Description

The "ShowTrendIndicator" property specifies whether the tendency (rising or falling) of the process value to be monitored is indicated by means of a small arrow.

Type

Bool

Access

Read-write

Syntax

```
Gauge.ShowTrendIndicator
```

See also

Gauge (Page 3300)

Gauge.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the gauge.

Type

String

Access

Read-only

Syntax

```
Gauge.StyleItemClass
```

See also

Gauge (Page 3300)

Gauge.TabIndex

Description

The "TabIndex" property returns the position of the gauge in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Gauge.TabIndex
```

See also

Gauge (Page 3300)

Gauge.ThresholdIndicators

Description

The "ThresholdIndicators" property specifies how parameterized limit values are visualized.

Type

Int32, HmiThresholdIndicator

Specifies the visualization:

- None (0): None
- Lines (1): Lines
- Markers (2): Markers

Access

Read-write

Syntax

```
Gauge.ThresholdIndicators
```

See also

Gauge (Page 3300)

Gauge.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the gauge.

Type

Object, HmiThresholdCollection (Page 3352)

Access

Read-only

Syntax

```
Gauge.Thresholds
```

See also

Gauge (Page 3300)

HmiThresholdCollection (Page 3352)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

Gauge.Thresholds (Page 3351)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 3352)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 3354)

See also

Threshold (Page 3354)

HmiThresholdCollection (Page 3352)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.
- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

See also

HmiThresholdCollection (Page 3352)

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 3354)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 3354)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 3354)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold
- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 3354)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 3354)

Gauge.Title

Description

The "Title" property specifies the caption that appears as the title.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Gauge.Title
```

See also

Gauge (Page 3300)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Gauge.Title (Page 3357)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 3357)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 3357)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 3357)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 3357)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 3357)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 3357)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Gauge.Title (Page 3357)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Gauge.Title (Page 3357)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Gauge.Title (Page 3357)

Gauge.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Gauge.ToolTipText
```

See also

Gauge (Page 3300)

Gauge.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Gauge.Top
```

See also

Gauge (Page 3300)

Gauge.TrendIndicatorColor

Description

The "TrendIndicatorColor" property specifies the color of the trend indicator. The trend indicator uses a small arrow to represent the tendency (rising or falling) of the process value to be monitored. To activate the trend indicator, the "ShowTrendIndicator" property must be activated.

Type

UInt32

Access

Read-write

Syntax

```
Gauge.TrendIndicatorColor
```

See also

Gauge (Page 3300)

Gauge.Visible

Description

The "Visible" property specifies whether the gauge is visible.

Type

Bool

Access

Read-write

Syntax

```
Gauge.Visible
```

See also

Gauge (Page 3300)

Gauge.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
Gauge.VisualizeQuality
```

See also

Gauge (Page 3300)

Gauge.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Gauge.Width
```

See also

Gauge (Page 3300)

Gauge.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the gauge.

Syntax

```
Gauge.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Gauge (Page 3300)

Gauge.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Gauge.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Gauge (Page 3300)

Gauge_OnActivated()

Description

The "OnActivated" event occurs when a gauge receives focus:

- A gauge is selected via the configured tab sequence.
- A gauge that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Gauge_OnActivated(item)
```

Context

`item`

Type: Object

Gauge where the event occurs.

See also

Gauge (Page 3300)

Gauge_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A gauge is right-clicked.
- A gauge is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Gauge_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Gauge where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Gauge (Page 3300)

Gauge_OnDeactivated()

Description

The "OnDeactivated" event occurs when the gauge loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Gauge_OnDeactivated(item)
```

Context

`item`

Type: Object

Gauge where the event occurs.

See also

Gauge (Page 3300)

Gauge_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the gauge is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Gauge_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Gauge where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Gauge (Page 3300)

Gauge_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the gauge is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Gauge_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Gauge where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Gauge (Page 3300)

Gauge_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A gauge is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a gauge has the focus.
- A gauge is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Gauge_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Gauge where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Gauge (Page 3300)

GraphicView

Description

The "GraphicView" object represents a graphic display in runtime.

Object type

HmiGraphicView

Properties

The "GraphicView" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the graphic display.
- **Enabled**
  Specifies whether the graphic display can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the graphic display is filled.
- **FillLevel**
  Specifies the fill of the graphic display in percent.
- **Graphic**
  Specifies the graphic.
- **GraphicStretchMode**
  Specifies the type of scaling of the graphic in the screen.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the graphic display is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the graphic display.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the graphic display is operable.
- **Padding**
  Specifies the distance of the graphic from the frame of the graphic display.
- **Parent**
  Returns the higher-level screen object.

- **RequireExplicitUnlock**
  Returns whether the graphic display is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the graphic display rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the graphic display is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the graphic display.
- **TabIndex**
  Returns the position of the graphic display in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the graphic display is visible.
- **Width**
  Specifies the width.

Methods

The "GraphicView" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the graphic display.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "GraphicView" object has the following events:

- **OnActivated()**
  Occurs when a graphic display receives focus.
- **OnContextTapped()**
  Occurs when a graphic display is right-clicked or long-touched.

- **OnDeactivated()**
  Occurs when a graphic display loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the graphic display is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the graphic display is in focus.
- **OnTapped()**
  Occurs when a graphic display is left-clicked or short-touched.

GraphicView.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
GraphicView.AlternateBackColor
```

See also

GraphicView (Page 3374)

GraphicView.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
GraphicView.Authorization
```

See also

GraphicView (Page 3374)

GraphicView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
GraphicView.BackColor
```

See also

GraphicView (Page 3374)

GraphicView.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe

- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
GraphicView.BackFillPattern
```

See also

GraphicView (Page 3374)

GraphicView.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the graphic display.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
GraphicView.CurrentQuality
```

See also

GraphicView (Page 3374)

GraphicView.Enabled

Description

The "Enabled" property specifies whether the graphic display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
GraphicView.Enabled
```

See also

GraphicView (Page 3374)

GraphicView.FillDirection

Description

The "FillDirection" property specifies the direction from which the graphic display is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom

- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
GraphicView.FillDirection
```

See also

GraphicView (Page 3374)

GraphicView.FillLevel

Description

The "FillLevel" property specifies the fill level of the graphic display in percent.

Type

UInt8

Access

Read-write

Syntax

```
GraphicView.FillLevel
```

See also

GraphicView (Page 3374)

GraphicView.Graphic

Description

The "Graphic" property specifies the graphic.

Type

String

Access

Read-write

Syntax

```
GraphicView.Graphic
```

See also

GraphicView (Page 3374)

GraphicView.GraphicStretchMode

Description

The "GraphicStretchMode" property specifies the type of scaling of the graphic in the screen.

Type

Int32, HmiGraphicStretchMode

Specifies the graphic scaling:

- None (0): The graphic is shown in its original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
GraphicView.GraphicStretchMode
```

See also

GraphicView (Page 3374)

GraphicView.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
GraphicView.Height
```

See also

GraphicView (Page 3374)

GraphicView.Layer

Description

The "Layer" property returns the layer of the screen in which the graphic display is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
GraphicView.Layer
```

See also

GraphicView (Page 3374)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

GraphicView.Layer (Page 3383)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

GraphicView.Layer (Page 3383)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

GraphicView.Layer (Page 3383)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

GraphicView.Layer (Page 3383)

GraphicView.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
GraphicView.Left
```

See also

GraphicView (Page 3374)

GraphicView.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
GraphicView.Margin
```

See also

GraphicView (Page 3374)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

GraphicView.Margin (Page 3386)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

GraphicView.Margin (Page 3386)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

GraphicView.Margin (Page 3386)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

GraphicView.Margin (Page 3386)

GraphicView.Name

Description

The "Name" property returns the name of the graphic display.

Type

String

Access

Read-only

Syntax

```
GraphicView.Name
```

See also

GraphicView (Page 3374)

GraphicView.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
GraphicView.Opacity
```

See also

GraphicView (Page 3374)

GraphicView.Operability

Description

The "Operability" property returns whether the graphic display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
GraphicView.Operability
```

See also

GraphicView (Page 3374)

GraphicView.Padding

Description

The "Padding" property specifies the distance of the graphic from the frame of the graphic display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
GraphicView.Padding
```

See also

GraphicView (Page 3374)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

GraphicView.Padding (Page 3390)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

GraphicView.Padding (Page 3390)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

GraphicView.Padding (Page 3390)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

GraphicView.Padding (Page 3390)

GraphicView.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
GraphicView.Parent
```

See also

GraphicView (Page 3374)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

GraphicView.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the graphic display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
GraphicView.RequireExplicitUnlock
```

See also

GraphicView (Page 3374)

GraphicView.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
GraphicView.RotationAngle
```

See also

GraphicView.RotationCenterX (Page 3395)

GraphicView.RotationCenterY (Page 3396)

GraphicView.RotationCenterPlacement (Page 3394)

GraphicView (Page 3374)

GraphicView.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the graphic display rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
GraphicView.RotationCenterPlacement
```

See also

GraphicView.RotationAngle (Page 3394)

GraphicView.RotationCenterX (Page 3395)

GraphicView.RotationCenterY (Page 3396)

GraphicView (Page 3374)

GraphicView.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
GraphicView.RotationCenterX
```

See also

GraphicView.RotationAngle (Page 3394)

GraphicView.RotationCenterY (Page 3396)

GraphicView.RotationCenterPlacement (Page 3394)

GraphicView (Page 3374)

GraphicView.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
GraphicView.RotationCenterY
```

See also

GraphicView.RotationAngle (Page 3394)

GraphicView.RotationCenterX (Page 3395)

GraphicView.RotationCenterPlacement (Page 3394)

GraphicView (Page 3374)

GraphicView.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
GraphicView.ShowFillLevel
```

See also

GraphicView (Page 3374)

GraphicView.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the graphic display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
GraphicView.ShowFocusVisual
```

See also

GraphicView (Page 3374)

GraphicView.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the graphic display.

Type

String

Access

Read-only

Syntax

```
GraphicView.StyleItemClass
```

See also

GraphicView (Page 3374)

GraphicView.TabIndex

Description

The "TabIndex" property returns the position of the graphic display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
GraphicView.TabIndex
```

See also

GraphicView (Page 3374)

GraphicView.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
GraphicView.ToolTipText
```

See also

GraphicView (Page 3374)

GraphicView.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
GraphicView.Top
```

See also

GraphicView (Page 3374)

GraphicView.Visible

Description

The "Visible" property specifies whether the graphic display is visible.

Type

Bool

Access

Read-write

Syntax

```
GraphicView.Visible
```

See also

GraphicView (Page 3374)

GraphicView.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
GraphicView.Width
```

See also

GraphicView (Page 3374)

GraphicView.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the graphic display.

Syntax

```
GraphicView.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

GraphicView (Page 3374)

GraphicView.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
GraphicView.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

GraphicView (Page 3374)

GraphicView_OnActivated()

Description

The "OnActivated" event occurs when a graphic display receives focus:

- A graphic display is selected via the configured tab sequence.
- A graphic display that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
GraphicView_OnActivated(item)
```

Context

item

Type: Object

Graphic display at which the event occurs.

See also

GraphicView (Page 3374)

GraphicView_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A graphic display is right-clicked.
- A graphic display is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
GraphicView_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Graphic display at which the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

GraphicView (Page 3374)

GraphicView_OnDeactivated()

Description

The "OnDeactivated" event occurs when the graphic display loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
GraphicView_OnDeactivated(item)
```

Context

`item`

Type: Object

Graphic display at which the event occurs.

See also

GraphicView (Page 3374)

GraphicView_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the graphic display is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
GraphicView_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Graphic display at which the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

GraphicView (Page 3374)

GraphicView_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the graphic display is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
GraphicView_OnKeyUp(item, keyCode, modifiers)
```

Context

`item`

Type:

Graphic display at which the event occurs.

`keyCode`

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

GraphicView (Page 3374)

GraphicView_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A graphic display is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a graphic display has the focus.
- A graphic display is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
GraphicView_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Graphic display at which the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

GraphicView (Page 3374)

Group

Description

The "Group" object represents grouped objects in runtime.

Object type

HmiGroup

Properties

The "Group" object has the following properties:

- **ContainedItems**
  Returns an array with the references of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the group.
- **Enabled**
  Specifies whether the group can be operated in runtime.
- **Height**
  Specifies the height.
- **Layer**
  Returns the screen layer in which the group is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the group.
- **Parent**
  Returns the higher-level screen object.
- **Properties**
  Allows access to the dynamic properties of the objects of the group.
- **RotationAngle**
  Specifies the rotation angle in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the group rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the group is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the group.
- **TabIndex**
  Returns the position of the group in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.

- **Visible**
  Specifies whether the group is visible.
- **Width**
  Specifies the width.

Methods

The "Group" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the group.
- **PropertyFlashing()**
  Configures flashing of a property.

Group.ContainedItems

Description

The "ContainedItems" property returns an array with the references of the grouped objects.

Type

ArrayOfSomRef

Access

Read-only

Syntax

```
Group.ContainedItems
```

See also

Group (Page 3408)

Group.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the group.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Group.CurrentQuality
```

See also

Group (Page 3408)

Group.Enabled

Description

The "Enabled" property specifies whether the group can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Group.Enabled
```

See also

Group (Page 3408)

Group.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Group.Height
```

See also

Group (Page 3408)

Group.Layer

Description

The "Layer" property returns the screen layer in which the group is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Group.Layer
```

See also

Group (Page 3408)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Group.Layer (Page 3412)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Group.Layer (Page 3412)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Group.Layer (Page 3412)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Group.Layer (Page 3412)

Group.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Group.Left
```

See also

Group (Page 3408)

Group.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Group.Margin
```

See also

Group (Page 3408)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Group.Margin (Page 3415)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Group.Margin (Page 3415)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Group.Margin (Page 3415)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Group.Margin (Page 3415)

Group.Name

Description

The "Name" property returns the name of the group.

Type

String

Access

Read-only

Syntax

```
Group.Name
```

See also

Group (Page 3408)

Group.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Group.Parent
```

See also

Group (Page 3408)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Group.Properties

Description

The "Properties" property allows access to the dynamic properties of the objects of the group.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
Group.Properties
```

See also

Group (Page 3408)

Group.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle in degrees.

Type

Int16

Access

Read-write

Syntax

```
Group.RotationAngle
```

See also

Group (Page 3408)

Group.RotationCenterPlacement (Page 3420)

Group.RotationCenterX (Page 3421)

Group.RotationCenterY (Page 3421)

Group.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the group rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in the DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Group.RotationCenterPlacement
```

See also

Group (Page 3408)

Group.RotationAngle (Page 3419)

Group.RotationCenterX (Page 3421)

Group.RotationCenterY (Page 3421)

Group.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Group.RotationCenterX
```

See also

Group (Page 3408)

Group.RotationAngle (Page 3419)

Group.RotationCenterPlacement (Page 3420)

Group.RotationCenterY (Page 3421)

Group.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Group.RotationCenterY
```

See also

Group (Page 3408)
Group.RotationAngle (Page 3419)
Group.RotationCenterPlacement (Page 3420)
Group.RotationCenterX (Page 3421)

Group.ShowFocusVisual

Description

The property "ShowFocusVisual" specifies whether the group is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Group.ShowFocusVisual
```

See also

Group (Page 3408)

Group.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the group.

Type

String

Access

Read-only

Syntax

```
Group.StyleItemClass
```

See also

Group (Page 3408)

Group.TabIndex

Description

The "TabIndex" property returns the position of the group in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Group.TabIndex
```

See also

Group (Page 3408)

Group.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Group.Top
```

See also

Group (Page 3408)

Group.Visible

Description

The "Visible" property specifies whether the group is visible.

Type

Bool

Access

Read-write

Syntax

```
Group.Visible
```

See also

Group (Page 3408)

Group.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Group.Width
```

See also

Group (Page 3408)

Group.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the group.

Syntax

```
Group.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Group (Page 3408)

Group.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Group.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Group (Page 3408)

IOField

Description

The "IOField" object represents an I/O field for the entry and display of process values in runtime.

Object type

HmiIOField

Properties

The "IOField" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the I/O field.
- **Enabled**
  Specifies whether the I/O field can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **InputBehavior**
  Specifies the input behavior.
- **IOFieldType**
  Specifies the mode of the I/O field.
- **Layer**
  Returns the layer of the screen in which the I/O field is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **MeasurementUnit**
  Returns the displayed unit.

- **MeasurementUnitType**
  Specifies the display format of the unit.
- **Name**
  Returns the name of the I/O field.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the I/O field is operable.
- **OutputFormat**
  Specifies the format for displaying the process values.
- **Padding**
  Specifies the distance of the content from the border of the I/O field.
- **Parent**
  Returns the higher-level screen object.
- **ProcessValue**
  Specifies the process value.
- **RenderingTemplate**
  Returns the name of the template from which the I/O field was created.
- **RequireExplicitUnlock**
  Returns whether the I/O field is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the I/O field rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the I/O field is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the I/O field.
- **TabIndex**
  Returns the position of the I/O field in the tab sequence.
- **TextTrimming**
  Specifies the type of text trimming if space is not sufficient.
- **Thresholds**
  Returns the list of all limit values of the I/O field.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.

- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the I/O field is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "IOField" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the I/O field.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "IOField" object has the following events:

- **OnActivated()**
  Occurs when an I/O field receives focus.
- **OnContextTapped()**
  Occurs when an I/O field is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when an I/O field loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the I/O field is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the I/O field is in focus.
- **OnTapped()**
  Occurs when an I/O field is left-clicked or short-touched.

IOField.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
IOField.AlternateBackColor
```

See also

IOField (Page 3426)

IOField.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
IOField.AlternateBorderColor
```

See also

IOField (Page 3426)

IOField.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
IOField.Authorization
```

See also

IOField (Page 3426)

IOField.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
IOField.BackColor
```

See also

IOField (Page 3426)

IOField.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
IOField.BorderColor
```

See also

IOField (Page 3426)

IOField.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
IOField.BorderWidth
```

See also

IOField (Page 3426)

IOField.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the I/O field.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.

- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
IOField.CurrentQuality
```

See also

IOField (Page 3426)

IOField.Enabled

Description

The "Enabled" property specifies whether the I/O field can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
IOField.Enabled
```

See also

IOField (Page 3426)

IOField.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
IOField.Font
```

See also

IOField (Page 3426)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

IOField.Font (Page 3433)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

IOField.Font (Page 3433)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

IOField.Font (Page 3433)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

IOField.Font (Page 3433)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

IOField.Font (Page 3433)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

IOField.Font (Page 3433)

IOField.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
IOField.ForeColor
```

See also

IOField (Page 3426)

IOField.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
IOField.Height
```

See also

IOField (Page 3426)

IOField.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
IOField.HorizontalTextAlignment
```

See also

IOField (Page 3426)

IOField.InputBehavior

Description

The "InputBehavior" property specifies the input behavior.

Type

Object, HmiInputBehaviorPart

Access

Read-write

Syntax

```
IOField.InputBehavior
```

See also

IOField (Page 3426)

InputBehavior.AcceptOnDeactivated

Description

The "AcceptOnDeactivated" property specifies whether the process value is written when the object loses the input focus.

Type

Bool

Access

Read-write

Syntax

```
InputBehavior.AcceptOnDeactivated
```

See also

IOField.InputBehavior (Page 3439)

InputBehavior.ClearOnActivate

Description

The "ClearOnActivate" property specifies whether the last value is deleted during input focus.

Type

Bool

Access

Read-write

Syntax

```
InputBehavior.ClearOnActivate
```

See also

IOField.InputBehavior (Page 3439)

InputBehavior.HiddenInput

Description

The "HiddenInput" property specifies whether the input value or an * is shown for each character during the input.

Type

Bool

Access

Read-write

Syntax

```
InputBehavior.HiddenInput
```

See also

IOField.InputBehavior (Page 3439)

InputBehavior.InputOnActivate

Description

The "InputOnActivate" property determines whether the input field accepts input when it receives focus.

Type

Bool

Access

Read-write

Syntax

```
InputBehavior.InputOnActivate
```

See also

IOField.InputBehavior (Page 3439)

IOField.IOFieldType

Description

The "IOFieldType" property specifies the mode of the I/O field.

Type

Int32, HmiIOFieldType

Specifies the type:

- Output (0): Output only
- InputOutput (2): Input and output

Access

Read-write

Syntax

```
IOField.IOFieldType
```

See also

IOField (Page 3426)

IOField.Layer

Description

The "Layer" property returns the layer of the screen in which the I/O field is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
IOField.Layer
```

See also

IOField (Page 3426)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

IOField.Layer (Page 3442)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

IOField.Layer (Page 3442)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

IOField.Layer (Page 3442)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

IOField.Layer (Page 3442)

IOField.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
IOField.Left
```

See also

IOField (Page 3426)

IOField.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
IOField.Margin
```

See also

IOField (Page 3426)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

IOField.Margin (Page 3445)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

IOField.Margin (Page 3445)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

IOField.Margin (Page 3445)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

IOField.Margin (Page 3445)

IOField.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
IOField.MeasurementUnit
```

See also

IOField (Page 3426)

IOField.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
IOField.MeasurementUnitType
```

See also

IOField (Page 3426)

IOField.Name

Description

The "Name" property returns the name of the I/O field.

Type

String

Access

Read-only

Syntax

```
IOField.Name
```

See also

IOField (Page 3426)

IOField.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
IOField.Opacity
```

See also

IOField (Page 3426)

IOField.Operability

Description

The "Operability" property returns whether the I/O field is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
IOField.Operability
```

See also

IOField (Page 3426)

IOField.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
IOField.OutputFormat
```

See also

IOField (Page 3426)

IOField.Padding

Description

The "Padding" property specifies the distance of the content from the border of the I/O field.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
IOField.Padding
```

See also

IOField (Page 3426)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

IOField.Padding (Page 3450)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

IOField.Padding (Page 3450)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

IOField.Padding (Page 3450)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

IOField.Padding (Page 3450)

IOField.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
IOField.Parent
```

See also

IOField (Page 3426)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

IOField.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
IOField.ProcessValue
```

See also

IOField (Page 3426)

IOField.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the I/O field was created.

Type

String

Access

Read-only

Syntax

```
IOField.RenderingTemplate
```

See also

IOField (Page 3426)

IOField.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the I/O field can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
IOField.RequireExplicitUnlock
```

See also

IOField (Page 3426)

IOField.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
IOField.RotationAngle
```

See also

IOField (Page 3426)

IOField.RotationCenterX (Page 3456)

IOField.RotationCenterY (Page 3457)

IOField.RotationCenterPlacement (Page 3456)

IOField.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the I/O field rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
IOField.RotationCenterPlacement
```

See also

IOField (Page 3426)

IOField.RotationAngle (Page 3455)

IOField.RotationCenterX (Page 3456)

IOField.RotationCenterY (Page 3457)

IOField.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
IOField.RotationCenterX
```

See also

IOField (Page 3426)

IOField.RotationAngle (Page 3455)

IOField.RotationCenterY (Page 3457)

IOField.RotationCenterPlacement (Page 3456)

IOField.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
IOField.RotationCenterY
```

See also

IOField (Page 3426)

IOField.RotationAngle (Page 3455)

IOField.RotationCenterX (Page 3456)

IOField.RotationCenterPlacement (Page 3456)

IOField.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the I/O field is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
IOField.ShowFocusVisual
```

See also

IOField (Page 3426)

IOField.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the I/O field.

Type

String

Access

Read-only

Syntax

```
IOField.StyleItemClass
```

See also

IOField (Page 3426)

IOField.TabIndex

Description

The "TabIndex" property returns the position of the I/O field in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
IOField.TabIndex
```

See also

IOField (Page 3426)

IOField.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
IOField.TextTrimming
```

See also

IOField (Page 3426)

IOField.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the I/O field.

Type

Object, HmiThresholdCollection (Page 3460)

Access

Read-only

Syntax

```
IOField.Thresholds
```

See also

IOField (Page 3426)

HmiThresholdCollection (Page 3460)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

IOField.Thresholds (Page 3460)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 3460)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 3462)

See also

HmiThresholdCollection (Page 3460)

Threshold (Page 3462)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.
- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

See also

HmiThresholdCollection (Page 3460)

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 3462)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 3462)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 3462)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold
- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 3462)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 3462)

IOField.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
IOField.ToolTipText
```

See also

IOField (Page 3426)

IOField.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
IOField.Top
```

See also

IOField (Page 3426)

IOField.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the text alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
IOField.VerticalTextAlignment
```

See also

IOField (Page 3426)

IOField.Visible

Description

The "Visible" property specifies whether the I/O field is visible.

Type

Bool

Access

Read-write

Syntax

```
IOField.Visible
```

See also

IOField (Page 3426)

IOField.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
IOField.VisualizeQuality
```

See also

IOField (Page 3426)

IOField.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
IOField.Width
```

See also

IOField (Page 3426)

IOField.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the I/O field.

Syntax

```
IOField.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

IOField (Page 3426)

IOField.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
IOField.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

IOField (Page 3426)

IOField_OnActivated()

Description

The "OnActivated" event occurs when an I/O field receives focus:

- An I/O field is selected via the configured tab sequence.
- An I/O field that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
IOField_OnActivated(item)
```

Context

item

Type: Object

I/O field where the event occurs.

See also

IOField (Page 3426)

IOField_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- An I/O field is right-clicked.
- An I/O field is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
IOField_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

I/O field where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

IOField (Page 3426)

IOField_OnDeactivated()

Description

The "OnDeactivated" event occurs when the I/O field loses focus when because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
IOField_OnDeactivated(item)
```

Context

item

Type: Object

I/O field where the event occurs.

See also

IOField (Page 3426)

IOField_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the I/O field is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
IOField_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

I/O field where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

IOField (Page 3426)

IOField_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the I/O field is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
IOField_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

I/O field where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

IOField (Page 3426)

IOField_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- An I/O field is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a I/O field has the focus.
- An I/O field is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
IOField_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

I/O field where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

IOField (Page 3426)

Label

Description

The "Label" object represents an editable text box.

Object type

HmiLabel

Properties

The "Label" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border width.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the editable text box.
- **Enabled**
  Specifies whether the editable text box can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height in the (Device Independent Unit).
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of a text.
- **Layer**
  Returns the screen layer in which the editable text box is located.
- **Left**
  Specifies the value of the X coordinate in the (Device Independent Unit).
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the editable text box.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the editable text box is operable.

- **Padding**
  Specifies the distance of the content from the border of the editable text box.
- **Parent**
  Returns the higher-level screen object (Parent container).
- **RenderingTemplate**
  Returns the name of the template from which the editable text box was created.
- **RequireExplicitUnlock**
  Returns whether the editable text box can only be operated while the associated button is being pressed.
- **RotationAngle**
  Specifies the rotation angle of the editable text box in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the editable text box rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the editable text box is highlighted when in focus.
- **StyleItemClass**
  Specifies the style which is applied to the editable text box.
- **TabIndex**
  Returns the position of the editable text box in the tab sequence.
- **Text**
  Specifies the label.
- **TextTrimming**
  Specifies the type of text trimming if space is insufficient.
- **TextWrapping**
  Specifies how text is wrapped if space is insufficient.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **VerticalTextAlignment**
  Specifies the vertical alignment of a text.
- **Visible**
  Specifies whether the editable text box is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width in (Device Independent Unit).

Methods

The "Label" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the editable text box.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Label" object has the following events:

- **OnActivated()**
  Occurs when the editable text box receives focus.
- **OnContextTapped()**
  Occurs when the editable text box is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when the editable text box loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the text box is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the text box is in focus.
- **OnTapped()**
  Occurs when the editable text box is left-clicked or short-touched.

Label.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Label.AlternateBackColor
```

See also

Label (Page 3476)

Label.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Label.AlternateBorderColor
```

See also

Label (Page 3476)

Label.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Label.Authorization
```

See also

Label (Page 3476)

Label.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Label.BackColor
```

See also

Label (Page 3476)

Label.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Label.BorderColor
```

See also

Label (Page 3476)

Label.BorderWidth

Description

The "BorderWidth" property specifies the border width.

Type

UInt32

Access

Read-write

Syntax

```
Label.BorderWidth
```

See also

Label (Page 3476)

Label.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the editable text box.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Label.CurrentQuality
```

See also

Label (Page 3476)

Label.Enabled

Description

The "Enabled" property specifies whether the editable text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Label.Enabled
```

See also

Label (Page 3476)

Label.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Label.Font
```

See also

Label (Page 3476)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Label.Font (Page 3483)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Label.Font (Page 3483)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Label.Font (Page 3483)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Label.Font (Page 3483)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Label.Font (Page 3483)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Label.Font (Page 3483)

Label.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Label.ForeColor
```

See also

Label (Page 3476)

Label.Height

Description

The "Height" property specifies the height in the (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Label.Height
```

See also

Label (Page 3476)

Label.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of a text.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Label.HorizontalTextAlignment
```

See also

Label (Page 3476)

Label.Layer

Description

The "Layer" property returns the screen layer in which the editable text box is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Label.Layer
```

See also

Label (Page 3476)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Label.Layer (Page 3489)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Label.Layer (Page 3489)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Label.Layer (Page 3489)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Label.Layer (Page 3489)

Label.Left

Description

The "Left" property sets the value of the X coordinate in the (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Label.Left
```

See also

Label (Page 3476)

Label.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Label.Margin
```

See also

Label (Page 3476)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Label.Margin (Page 3492)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Label.Margin (Page 3492)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Label.Margin (Page 3492)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Label.Margin (Page 3492)

Label.Name

Description

The "Name" property returns the name of the editable text box.

Type

String

Access

Read-only

Syntax

```
Label.Name
```

See also

Label (Page 3476)

Label.Opacity

Description

The "Opacity" property specifies the opacity. The "0" value indicates completely transparency.

Type

Float

Access

Read-write

Syntax

```
Label.Opacity
```

See also

Label (Page 3476)

Label.Operability

Description

The "Operability" property returns whether the editable text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Label.Operability
```

See also

Label (Page 3476)

Label.Padding

Description

The "Padding" property specifies the distance of the content from the border of the editable text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
Label.Padding
```

See also

Label (Page 3476)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

Label.Padding (Page 3496)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

Label.Padding (Page 3496)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

Label.Padding (Page 3496)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

Label.Padding (Page 3496)

Label.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Label.Parent
```

See also

Label (Page 3476)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Label.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the editable text box was created.

Type

String

Access

Read-only

Syntax

```
Label.RenderingTemplate
```

See also

Label (Page 3476)

Label.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the editable text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Label.RequireExplicitUnlock
```

See also

Label (Page 3476)

Label.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle of the editable text box in degrees.

Type

Int16

Access

Read-write

Syntax

```
Label.RotationAngle
```

See also

Label (Page 3476)

Label.RotationCenterPlacement (Page 3500)

Label.RotationCenterX (Page 3501)

Label.RotationCenterY (Page 3502)

Label.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the editable text box rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in the DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Label.RotationCenterPlacement
```

See also

Label (Page 3476)

Label.RotationAngle (Page 3500)

Label.RotationCenterX (Page 3501)

Label.RotationCenterY (Page 3502)

Label.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point.

Type

Float

Access

Read-write

Syntax

```
Label.RotationCenterX
```

See also

Label (Page 3476)

Label.RotationAngle (Page 3500)

Label.RotationCenterPlacement (Page 3500)

Label.RotationCenterY (Page 3502)

Label.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point.

Type

Float

Access

Read-write

Syntax

```
Label.RotationCenterY
```

See also

Label (Page 3476)

Label.RotationAngle (Page 3500)

Label.RotationCenterPlacement (Page 3500)

Label.RotationCenterX (Page 3501)

Label.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the editable text box is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Label.ShowFocusVisual
```

See also

Label (Page 3476)

Label.StyleItemClass

Description

The "StyleItemClass" property specifies the style which is applied to the editable text box.

Type

String

Access

Read-only

Syntax

```
Label.StyleItemClass
```

See also

Label (Page 3476)

Label.TabIndex

Description

The "TabIndex" property returns the position of the editable text box in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Label.TabIndex
```

See also

Label (Page 3476)

Label.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Label.Text
```

See also

Label (Page 3476)

Label.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Label.TextTrimming
```

See also

Label (Page 3476)

Label.TextWrapping

Description

The "TextWrapping" property specifies how text is wrapped if there is insufficient space.

Type

Int32, HmiTextWrapping

Specifies the text break:

- NoWrap (0): No text break
- WordWrap (1): Text break at the end of the line

Access

Read-write

Syntax

```
Label.TextWrapping
```

See also

Label (Page 3476)

Label.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Label.ToolTipText
```

See also

Label (Page 3476)

Label.Top

Description

The "Top" property specifies the value of the Y coordinate.

Type

Int32

Access

Read-write

Syntax

```
Label.Top
```

See also

Label (Page 3476)

Label.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of a text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Label.VerticalTextAlignment
```

See also

Label (Page 3476)

Label.Visible

Description

The "Visible" property specifies whether the editable text box is visible.

Type

Bool

Access

Read-write

Syntax

```
Label.Visible
```

See also

Label (Page 3476)

Label.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
Label.VisualizeQuality
```

See also

Label (Page 3476)

Label.Width

Description

The "Width" property specifies the width in the (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Label.Width
```

See also

Label (Page 3476)

Label.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the editable text box.

Syntax

```
Label.CheckAuthorization()
```

Parameters

-

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Label (Page 3476)

Label.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Label.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Label (Page 3476)

Label_OnActivated()

Description

The "OnActivated" event occurs when an editable text box receives focus:

- A text box is selected via the configured tab sequence.
- A text box that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Label_OnActivated(item)
```

Context

item

Type: Object

Text box where the event occurs.

See also

Label (Page 3476)

Label_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A text box is right-clicked.
- A text box is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Label_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Text box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Label (Page 3476)

Label_OnDeactivated()

Description

The "OnDeactivated" event occurs when the text box loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Label_OnDeactivated(item)
```

Context

item

Type: Object

Text box where the event occurs.

See also

Label (Page 3476)

Label_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the text box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Label_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Text box where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Label (Page 3476)

Label_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the text box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Label_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Text box where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Label (Page 3476)

Label_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A text box is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a text box has the focus.
- A text box is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Label_OnTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Text box where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Label (Page 3476)

Line

Description

The "Line" object represents a line in runtime.

Object type

HmiLine

Properties

The "Line" object has the following properties:

- **AlternateLineColor**
  Specifies the second line color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **CapType**
  Specifies the shape of the line ends.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the line.
- **DashType**
  Specifies the stroke style of the line.
- **Enabled**
  Specifies whether the line can be operated in runtime.
- **EndType**
  Specifies the type of line end.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the line is located.
- **Left**
  Specifies the value of the X coordinate.
- **LineColor**
  Specifies the line color.
- **LineWidth**
  Specifies the line thickness.
- **Margin**
  Specifies the margin.

- **Name**
  Returns the name of the line.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the line is operable.
- **Parent**
  Returns the higher-level screen object.
- **RequireExplicitUnlock**
  Returns whether the line is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the line rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the line is highlighted when in focus.
- **StartType**
  Specifies the type of line start.
- **StyleItemClass**
  Returns the style which is applied to the line.
- **TabIndex**
  Returns the position of the line in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the line is visible.
- **Width**
  Specifies the width.
- **X1**
  Specifies the X coordinate of the starting point of the line.
- **X2**
  Specifies the X coordinate of the end point of the line.
- **Y1**
  Specifies the Y coordinate of the starting point of the line.
- **Y2**
  Specifies the Y coordinate of the end point of the line.

Methods

The "Line" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the line.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Line" object has the following events:

- **OnActivated()**
  Occurs when a line receives focus.
- **OnContextTapped()**
  Occurs when a line is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a line loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the line is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the line is in focus.
- **OnTapped()**
  Occurs when a line is left-clicked or short-touched.

Line.AlternateLineColor

Description

The "AlternateLineColor" property specifies the second line color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Line.AlternateLineColor
```

See also

Line (Page 3517)

Line.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Line.Authorization
```

See also

Line (Page 3517)

Line.CapType

Description

The "CapType" property specifies the shape of the line ends.

Type

Int32, HmiCapType

Specifies the line ends:

- Round (0): Round (line extends beyond the line end point with half the line thickness)
- Square (256): Square (line extends beyond the line end point with half the line thickness)
- Flat (512): Justified (line ends at the line end point)

Access

Read-write

Syntax

```
Line.CapType
```

See also

Line (Page 3517)

Line.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the line.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Line.CurrentQuality
```

See also

Line (Page 3517)

Line.DashType

Description

The "DashType" property specifies the stroke style of the line.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Line.DashType
```

See also

Line (Page 3517)

Line.Enabled

Description

The "Enabled" property specifies whether the line can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Line.Enabled
```

See also

Line (Page 3517)

Line.EndType

Description

The "EndType" property specifies the line end type.

Type

Int32, HmiLineEndType

Specifies the line end type:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reverse arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
Line.EndType
```

See also

Line (Page 3517)

Line.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Line.Height
```

See also

Line (Page 3517)

Line.Layer

Description

The "Layer" property returns the layer of the screen in which the line is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Line.Layer
```

See also

Line (Page 3517)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Line.Layer (Page 3524)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Line.Layer (Page 3524)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Line.Layer (Page 3524)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Line.Layer (Page 3524)

Line.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Line.Left
```

See also

Line (Page 3517)

Line.LineColor

Description

The "LineColor" property specifies the line color.

Type

UInt32

Access

Read-write

Syntax

```
Line.LineColor
```

See also

Line (Page 3517)

Line.LineWidth

Description

The "LineWidth" property specifies the line thickness.

Type

UInt8

Access

Read-write

Syntax

```
Line.LineWidth
```

See also

Line (Page 3517)

Line.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Line.Margin
```

See also

Line (Page 3517)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Line.Margin (Page 3528)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Line.Margin (Page 3528)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Line.Margin (Page 3528)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Line.Margin (Page 3528)

Line.Name

Description

The "Name" property returns the name of the line.

Type

String

Access

Read-only

Syntax

```
Line.Name
```

See also

Line (Page 3517)

Line.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Line.Opacity
```

See also

Line (Page 3517)

Line.Operability

Description

The "Operability" property returns whether the line is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Line.Operability
```

See also

Line (Page 3517)

Line.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Line.Parent
```

See also

Line (Page 3517)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Line.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the line can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Line.RequireExplicitUnlock
```

See also

Line (Page 3517)

Line.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Line.RotationAngle
```

See also

Line (Page 3517)

Line.RotationCenterPlacement (Page 3533)

Line.RotationCenterX (Page 3534)

Line.RotationCenterY (Page 3535)

Line.RotationCenterPlacement

Description

The "RotationCenterPlacement" property specifies the reference point around which the line rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Line.RotationCenterPlacement
```

See also

Line (Page 3517)

Line.RotationAngle (Page 3533)

Line.RotationCenterX (Page 3534)

Line.RotationCenterY (Page 3535)

Line.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Line.RotationCenterX
```

See also

Line (Page 3517)

Line.RotationAngle (Page 3533)

Line.RotationCenterPlacement (Page 3533)

Line.RotationCenterY (Page 3535)

Line.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Line.RotationCenterY
```

See also

Line (Page 3517)

Line.RotationAngle (Page 3533)

Line.RotationCenterPlacement (Page 3533)

Line.RotationCenterX (Page 3534)

Line.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the line is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Line.ShowFocusVisual
```

See also

Line (Page 3517)

Line.StartType

Description

The "StartType" property specifies the type of line start.

Type

Int32, HmiLineEndType

Specifies the start of the line:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reverse arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
Line.StartType
```

See also

Line (Page 3517)

Line.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the line.

Type

String

Access

Read-only

Syntax

```
Line.StyleItemClass
```

See also

Line (Page 3517)

Line.TabIndex

Description

The "TabIndex" property returns the position of the line in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Line.TabIndex
```

See also

Line (Page 3517)

Line.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Line.ToolTipText
```

See also

Line (Page 3517)

Line.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Line.Top
```

See also

Line (Page 3517)

Line.Visible

Description

The "Visible" property specifies whether the line is visible.

Type

Bool

Access

Read-write

Syntax

```
Line.Visible
```

See also

Line (Page 3517)

Line.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Line.Width
```

See also

Line (Page 3517)

Line.X1

Description

The "X1" property specifies the X coordinate of the starting point of the line.

Type

Int32

Access

Read-write

Syntax

```
Line.X1
```

See also

Line (Page 3517)
Line.X2 (Page 3540)
Line.Y1 (Page 3541)

Line.X2

Description

The "X2" property specifies the X coordinate of the end point of the line.

Type

Int32

Access

Read-write

Syntax

```
Line.X2
```

See also

Line (Page 3517)
Line.X1 (Page 3540)
Line.Y2 (Page 3541)

Line.Y1

Description

The "Y1" property specifies the Y coordinate of the starting point of the line.

Type

Int32

Access

Read-write

Syntax

```
Line.Y1
```

See also

Line (Page 3517)
Line.X1 (Page 3540)
Line.Y2 (Page 3541)

Line.Y2

Description

The "Y2" property specifies the Y coordinate of the end point of the line.

Type

Int32

Access

Read-write

Syntax

```
Line.Y2
```

See also

Line (Page 3517)

Line.X2 (Page 3540)

Line.Y1 (Page 3541)

Line.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the line.

Syntax

```
Line.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Line (Page 3517)

Line.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Line.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Line (Page 3517)

Line_OnActivated()

Description

The "OnActivated" event occurs when a line receives focus:

- A line is selected via the configured tab sequence.
- A line that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Line_OnActivated(item)
```

Context

item

Type: Object

Line where the event occurs.

See also

Line (Page 3517)

Line_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A line is right-clicked.
- A line is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Line_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Line where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Line (Page 3517)

Line_OnDeactivated()

Description

The "OnDeactivated" event occurs when the line loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Line_OnDeactivated(item)
```

Context

item

Type: Object

Line where the event occurs.

See also

Line (Page 3517)

Line_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the line is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Line_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Line where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Line (Page 3517)

Line_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while a line is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Line_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Line where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Line (Page 3517)

Line_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A line is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a line has the focus.
- A line is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Line_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Line where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Line (Page 3517)

ListBox

Description

The "ListBox" object represents a list box in runtime.

Object type

HmiListBox

Properties

The "ListBox" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **Content**
  Specifies the display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the list box.
- **Enabled**
  Specifies whether the list box can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the list box is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the list box.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the list box is operable.
- **Padding**
  Specifies the distance of the content from the border of the list box.
- **Parent**
  Returns the higher-level screen object.

- **ProcessValue**
  Specifies the process value.
- **RenderingTemplate**
  Returns the name of the template from which the list box was created.
- **RequireExplicitUnlock**
  Returns whether the list box is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the list box rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **SelectionItemHeight**
  Specifies the height of the list entries.
- **SelectionItems**
  Returns the list of all list entries of the list box.
- **SelectionMode**
  Specifies whether one or more list entries can be selected from the list box.
- **SelectorPosition**
  Specifies the horizontal alignment of the list entries.
- **ShowFocusVisual**
  Specifies whether the list box is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the list box.
- **TabIndex**
  Returns the position of the list box in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the list box is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "ListBox" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the list box.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ListBox" object has the following events:

- **OnActivated()**
  Occurs when a list box receives focus.
- **OnContextTapped()**
  Occurs when a list box is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a list box loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the list box is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the list box is in focus.
- **OnTapped()**
  Occurs when a list box is left-clicked or short-touched.

ListBox.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ListBox.AlternateBackColor
```

See also

ListBox (Page 3550)

ListBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ListBox.AlternateBorderColor
```

See also

ListBox (Page 3550)

ListBox.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ListBox.Authorization
```

See also

ListBox (Page 3550)

ListBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ListBox.BackColor
```

See also

ListBox (Page 3550)

ListBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ListBox.BorderColor
```

See also

ListBox (Page 3550)

ListBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ListBox.BorderWidth
```

See also

ListBox (Page 3550)

ListBox.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ListBox.Content
```

See also

ListBox (Page 3550)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ListBox.Content (Page 3556)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ListBox.Content (Page 3556)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ListBox.Content (Page 3556)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ListBox.Content (Page 3556)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ListBox.Content (Page 3556)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ListBox.Content (Page 3556)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ListBox.Content (Page 3556)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ListBox.Content (Page 3556)

ListBox.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the list box.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ListBox.CurrentQuality
```

See also

ListBox (Page 3550)

ListBox.Enabled

Description

The "Enabled" property specifies whether the list box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ListBox.Enabled
```

See also

ListBox (Page 3550)

ListBox.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ListBox.Font
```

See also

ListBox (Page 3550)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ListBox.Font (Page 3563)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ListBox.Font (Page 3563)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ListBox.Font (Page 3563)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ListBox.Font (Page 3563)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ListBox.Font (Page 3563)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ListBox.Font (Page 3563)

ListBox.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ListBox.ForeColor
```

See also

ListBox (Page 3550)

ListBox.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ListBox.Height
```

See also

ListBox (Page 3550)

ListBox.Layer

Description

The "Layer" property returns the layer of the screen in which the list box is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ListBox.Layer
```

See also

ListBox (Page 3550)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ListBox.Layer (Page 3567)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ListBox.Layer (Page 3567)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ListBox.Layer (Page 3567)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ListBox.Layer (Page 3567)

ListBox.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ListBox.Left
```

See also

ListBox (Page 3550)

ListBox.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ListBox.Margin
```

See also

ListBox (Page 3550)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ListBox.Margin (Page 3570)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ListBox.Margin (Page 3570)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ListBox.Margin (Page 3570)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ListBox.Margin (Page 3570)

ListBox.Name

Description

The "Name" property returns the name of the list box.

Type

String

Access

Read-only

Syntax

```
ListBox.Name
```

See also

ListBox (Page 3550)

ListBox.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
ListBox.Opacity
```

See also

ListBox (Page 3550)

ListBox.Operability

Description

The "Operability" property returns whether the list box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ListBox.Operability
```

See also

ListBox (Page 3550)

ListBox.Padding

Description

The "Padding" property specifies the distance of the content from the border of the list box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ListBox.Padding
```

See also

ListBox (Page 3550)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ListBox.Padding (Page 3574)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ListBox.Padding (Page 3574)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ListBox.Padding (Page 3574)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ListBox.Padding (Page 3574)

ListBox.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ListBox.Parent
```

See also

ListBox (Page 3550)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ListBox.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
ListBox.ProcessValue
```

See also

ListBox (Page 3550)

ListBox.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the list box was created.

Type

String

Access

Read-only

Syntax

```
ListBox.RenderingTemplate
```

See also

ListBox (Page 3550)

ListBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the list box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ListBox.RequireExplicitUnlock
```

See also

ListBox (Page 3550)

ListBox.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
ListBox.RotationAngle
```

See also

ListBox (Page 3550)

ListBox.RotationCenterPlacement (Page 3580)

ListBox.RotationCenterX (Page 3580)

ListBox.RotationCenterY (Page 3581)

ListBox.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the list box rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
ListBox.RotationCenterPlacement
```

See also

ListBox (Page 3550)

ListBox.RotationAngle (Page 3579)

ListBox.RotationCenterX (Page 3580)

ListBox.RotationCenterY (Page 3581)

ListBox.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
ListBox.RotationCenterX
```

See also

ListBox (Page 3550)

ListBox.RotationAngle (Page 3579)

ListBox.RotationCenterPlacement (Page 3580)

ListBox.RotationCenterY (Page 3581)

ListBox.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
ListBox.RotationCenterY
```

See also

ListBox (Page 3550)

ListBox.RotationAngle (Page 3579)

ListBox.RotationCenterPlacement (Page 3580)

ListBox.RotationCenterX (Page 3580)

ListBox.SelectionItemHeight

Description

The "SelectionItemHeight" property specifies the height of the list entries. The value "0" indicates that the height is calculated automatically.

Type

UInt16

Access

Read-write

Syntax

```
ListBox.SelectionItemHeight
```

See also

ListBox (Page 3550)

ListBox.SelectionItems

Description

The "SelectionItems" property returns the list of all list entries ("SelectionItem" objects) of the list box.

Type

Object, HmiSelectionItemCollection (Page 3583)

Access

Read-only

Syntax

```
ListBox.SelectionItems
```

See also

ListBox (Page 3550)

HmiSelectionItemCollection (Page 3583)

HmiSelectionItemCollection

Description

The "HmiSelectionItemCollection" object is a list of all entries ("SelectionItem" objects) of a list object.

Use

The "HmiSelectionItemCollection" object is a list and can be counted and enumerated. You can access the "HmiSelectionItemCollection" list using the index or the tag name.

Object type

HmiSelectionItemCollection

Properties

The "HmiSelectionItemCollection" object has the following properties:

- **Count**
  Returns the number of list entries of the "HmiSelectionItemCollection" list.

Methods

The "HmiSelectionItemCollection" object has the following methods:

- **Item()**
  Returns a list entry of the "HmiSelectionItemCollection" list.

See also

ListBox.SelectionItems (Page 3582)

HmiSelectionItemCollection.Count

Description

The "Count" property returns the number of list entries in the "HmiSelectionItemCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiSelectionItemCollection.Count
```

See also

HmiSelectionItemCollection (Page 3583)

HmiSelectionItemCollection.Item()

Description

The "Item" method returns a list entry of the "HmiSelectionItemCollection" list.

Syntax

```
HmiSelectionItemCollection[.Item](HmiSelectionItemName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiSelectionItemCollection" object.

Parameters

`HmiSelectionItemName`

Type: String

Name of the list entry

Return value

Object, HmiSelectionItemPart (Page 3584)

See also

HmiSelectionItemCollection (Page 3583)

SelectionItem (Page 3584)

SelectionItem

Description

The "SelectionItem" object represents a list entry.

Object type

HmiSelectionItemPart

Properties

The "SelectionItem" object has the following properties:

- **Graphic**
  Specifies the graphic of the list entry.
- **IsSelected**
  Specifies whether the list entry is selected.
- **Text**
  Specifies the list entry text.

Methods

--

See also

HmiSelectionItemCollection (Page 3583)

SelectionItem.Graphic

Description

The "Graphic" property specifies the graphic of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Graphic
```

See also

SelectionItem (Page 3584)

SelectionItem.IsSelected

Description

The "IsSelected" property specifies whether the list entry is selected.

Type

Bool

Access

Read-write

Syntax

```
SelectionItem.IsSelected
```

See also

SelectionItem (Page 3584)

SelectionItem.Text

Description

The "Text" property specifies the text of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Text
```

See also

SelectionItem (Page 3584)

ListBox.SelectionMode

Description

The "SelectionMode" property specifies whether one or more list entries can be selected in a list box.

Type

Int32, HmiSelectionMode

Specifies the selection type:

- NonExclusive (0): Selection of multiple list entries possible
- Exclusive (1): Selection of only one list entry possible

Access

Read-write

Syntax

```
ListBox.SelectionMode
```

See also

ListBox (Page 3550)

ListBox.SelectorPosition

Description

The "SelectorPosition" property specifies the horizontal alignment of the list entries.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
ListBox.SelectorPosition
```

See also

ListBox (Page 3550)

ListBox.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the list box is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ListBox.ShowFocusVisual
```

See also

ListBox (Page 3550)

ListBox.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the list box.

Type

String

Access

Read-only

Syntax

```
ListBox.StyleItemClass
```

See also

ListBox (Page 3550)

ListBox.TabIndex

Description

The "TabIndex" property returns the position of the list box in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ListBox.TabIndex
```

See also

ListBox (Page 3550)

ListBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
ListBox.ToolTipText
```

See also

ListBox (Page 3550)

ListBox.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ListBox.Top
```

See also

ListBox (Page 3550)

ListBox.Visible

Description

The "Visible" property specifies whether the list box is visible.

Type

Bool

Access

Read-write

Syntax

```
ListBox.Visible
```

See also

ListBox (Page 3550)

ListBox.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
ListBox.VisualizeQuality
```

See also

ListBox (Page 3550)

ListBox.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ListBox.Width
```

See also

ListBox (Page 3550)

ListBox.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the list box.

Syntax

```
ListBox.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ListBox (Page 3550)

ListBox.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ListBox.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ListBox (Page 3550)

ListBox_OnActivated()

Description

The "OnActivated" event occurs when a list box receives focus:

- A list box is selected via the configured tab sequence.
- A list box that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
ListBox_OnActivated(item)
```

Context

`item`

Type: Object

List box where the event occurs.

See also

ListBox (Page 3550)

ListBox_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A list box is right-clicked.
- A list box is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
ListBox_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

List box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ListBox (Page 3550)

ListBox_OnDeactivated()

Description

The "OnDeactivated" event occurs when the list box loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
ListBox_OnDeactivated(item)
```

Context

item

Type: Object

List box where the event occurs.

See also

ListBox (Page 3550)

ListBox_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the list box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
ListBox_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

List box where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

ListBox (Page 3550)

ListBox_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the list box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
ListBox_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

List box where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

ListBox (Page 3550)

ListBox_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A list box is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a list box has the focus.
- A list box is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
ListBox_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

List box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ListBox (Page 3550)

MediaControl

Description

The "MediaControl" object represents a Media Player for playback in runtime.

Object type

HmiMediaControl

Properties

The "MediaControl" object has the following properties:

- **AutoPlay**
  Specifies whether Autoplay is activated.
- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the Media Player.
- **Enabled**
  Specifies whether the Media Player can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the Media Player.
- **Layer**
  Returns the screen layer in which the Media Player is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the Media Player.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the Media Player was created.
- **ShowFocusVisual**
  Specifies whether the Media Player is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the Media Player.
- **StyleItemClass**
  Specifies the information bar of the Media Player.
- **TabIndex**
  Returns the position of the Media Player in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the Media Player.
- **Top**
  Specifies the value of the Y coordinate.

- **Url**
  Specifies the URL played by the Media Player.
- **VideoOutput**
  Specifies the scaling of the video output.
- **Visible**
  Specifies whether the Media Player is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the Media Player.

Methods

The "MediaControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the Media Player.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **Pause()**
  Pauses playback.
- **Play()**
  Starts or resumes playback.
- **PropertyFlashing()**
  Configures flashing of a property.
- **Stop()**
  Stops playback.

Events

The "MediaControl" object has the following events:

- **OnActivated()**
  Occurs when a Media Player receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the Media Player.
- **OnDeactivated()**
  Occurs when a Media Player loses focus.
- **OnInitialized()**
  Occurs when a Media Player has been successfully initialized and the data connection to the PLC has been established.
- **OnPaused()**
  Occurs when playback is paused in the Media Player.

- **OnPlaying()**
  Occurs when playback is started or resumed in the Media Player.
- **OnStopped()**
  Occurs when playback is stopped in the Media Player.

MediaControl.AutoPlay

Description

The "AutoPlay" property specifies whether Autoplay is activated.

Type

Bool

Access

Read-write

Syntax

```
MediaControl.AutoPlay
```

See also

MediaControl (Page 3600)

MediaControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
MediaControl.BackColor
```

See also

MediaControl (Page 3600)

MediaControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
MediaControl.Caption
```

See also

MediaControl (Page 3600)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

MediaControl.Caption (Page 3604)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 3604)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 3604)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 3604)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 3604)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 3604)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 3604)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

MediaControl.Caption (Page 3604)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

MediaControl.Caption (Page 3604)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

MediaControl.Caption (Page 3604)

MediaControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
MediaControl.CaptionColor
```

See also

MediaControl (Page 3600)

MediaControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the Media Player.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
MediaControl.CurrentQuality
```

See also

MediaControl (Page 3600)

MediaControl.Enabled

Description

The "Enabled" property specifies whether the Media Player can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
MediaControl.Enabled
```

See also

MediaControl (Page 3600)

MediaControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
MediaControl.Height
```

See also

MediaControl (Page 3600)

MediaControl.Icon

Description

The "Icon" property specifies the icon of the Media Player.

Type

String

Access

Read-write

Syntax

```
MediaControl.Icon
```

See also

MediaControl (Page 3600)

MediaControl.Layer

Description

The "Layer" property returns the screen layer in which the Media Player is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
MediaControl.Layer
```

See also

MediaControl (Page 3600)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

MediaControl.Layer (Page 3612)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

MediaControl.Layer (Page 3612)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

MediaControl.Layer (Page 3612)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

MediaControl.Layer (Page 3612)

MediaControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
MediaControl.Left
```

See also

MediaControl (Page 3600)

MediaControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
MediaControl.Margin
```

See also

MediaControl (Page 3600)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

MediaControl.Margin (Page 3615)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

MediaControl.Margin (Page 3615)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

MediaControl.Margin (Page 3615)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

MediaControl.Margin (Page 3615)

MediaControl.Name

Description

The "Name" property returns the name of the Media Player.

Type

String

Access

Read-only

Syntax

```
MediaControl.Name
```

See also

MediaControl (Page 3600)

MediaControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
MediaControl.Parent
```

See also

MediaControl (Page 3600)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

MediaControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the Media Player was created.

Type

String

Access

Read-only

Syntax

```
MediaControl.RenderingTemplate
```

See also

MediaControl (Page 3600)

MediaControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the Media Player is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
MediaControl.ShowFocusVisual
```

See also

MediaControl (Page 3600)

MediaControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the Media Player.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
MediaControl.StatusBar
```

See also

MediaControl (Page 3600)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

MediaControl.StatusBar (Page 3620)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 3621)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

MediaControl.StatusBar (Page 3620)

HmiControlBarElementCollection (Page 3621)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 3621)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 3621)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 3638)

See also

HmiControlBarElementCollection (Page 3621)

Control Bar Elements (Page 3638)

Control Bar Elements

Description

Control Bar Elements (Page 3638)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

MediaControl.StatusBar (Page 3620)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

MediaControl.StatusBar (Page 3620)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 3624)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 3624)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 3624)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 3624)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 3624)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 3624)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

MediaControl.StatusBar (Page 3620)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 3628)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 3628)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 3628)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 3628)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

MediaControl.StatusBar (Page 3620)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 3630)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 3630)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 3630)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 3630)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

MediaControl.StatusBar (Page 3620)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

MediaControl.StatusBar (Page 3620)

MediaControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the Media Player.

Type

String

Access

Read-only

Syntax

```
MediaControl.StyleItemClass
```

See also

MediaControl (Page 3600)

MediaControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the Media Player.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
MediaControl.ToolBar
```

See also

MediaControl (Page 3600)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

MediaControl.ToolBar (Page 3634)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 3636)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

MediaControl.ToolBar (Page 3634)

HmiControlBarElementCollection (Page 3636)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 3635)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 3636)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 3638)

See also

HmiControlBarElementCollection (Page 3636)

Control Bar Elements (Page 3638)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.

- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 3638)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 3638)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 3638)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 3638)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 3638)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 3643)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 3643)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 3643)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 3643)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 3643)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above

- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 3643)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 3643)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 3643)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 3638)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 3638)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog

- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled

- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 3638)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 3653)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 3653)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 3653)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 3653)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 3638)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 3638)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 3638)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 3638)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 3659)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 3659)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 3659)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 3659)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 3638)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 3638)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 3638)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 3664)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 3666)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 3666)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 3666)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 3666)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 3666)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 3666)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 3666)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 3666)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 3664)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 3676)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 3676)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 3676)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 3676)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 3664)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 3681)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 3681)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 3681)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 3681)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 3664)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 3664)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 3686)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 3693)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 3693)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 3693)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 3693)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 3686)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 3698)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 3698)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 3698)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 3698)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 3686)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 3686)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 3709)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 3709)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 3709)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 3709)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 3714)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 3714)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 3714)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 3714)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 3703)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 3703)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.

- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms

- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 3719)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 3728)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 3728)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 3728)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 3728)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 3719)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 3733)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 3733)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 3733)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 3733)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 3719)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 3719)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.

- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 3739)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 3745)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 3745)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 3739)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 3756)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 3756)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 3756)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 3756)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 3739)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 3761)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 3761)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 3761)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 3761)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 3739)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 3739)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

MediaControl.ToolBar (Page 3634)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

MediaControl.ToolBar (Page 3634)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 3767)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 3767)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 3767)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 3767)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 3767)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 3767)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

MediaControl.ToolBar (Page 3634)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 3771)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 3771)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 3771)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 3771)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

MediaControl.ToolBar (Page 3634)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 3773)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 3773)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 3773)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 3773)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

MediaControl.ToolBar (Page 3634)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

MediaControl.ToolBar (Page 3634)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

MediaControl.ToolBar (Page 3634)

MediaControl.TabIndex

Description

The "TabIndex" property returns the position of the Media Player in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
MediaControl.TabIndex
```

See also

MediaControl (Page 3600)

MediaControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
MediaControl.Top
```

See also

MediaControl (Page 3600)

MediaControl.Url

Description

The "Url" property specifies the URL played by the Media Player.

Type

String

Access

Read-write

Syntax

```
MediaControl.Url
```

See also

MediaControl (Page 3600)

MediaControl.VideoOutput

Description

The "VideoOutput" property specifies the scaling of the video output.

Type

Int32, HmiVideoOutput

Specifies the scaling:

- Undefined (0): Not defined.
- Stretch (1): The video is scaled to match.
- PreserveAspectFit (2): The video is scaled uniformly so that it fits without being cropped.
- PreserveAspectCrop (3): The video is scaled uniformly so that it fits, with cropping if required.

Access

Read-write

Syntax

```
MediaControl.VideoOutput
```

See also

MediaControl (Page 3600)

MediaControl.Visible

Description

The "Visible" property specifies whether the Media Player is visible.

Type

Bool

Access

Read-write

Syntax

```
MediaControl.Visible
```

See also

MediaControl (Page 3600)

MediaControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
MediaControl.Width
```

See also

MediaControl (Page 3600)

MediaControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the Media Player.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized

- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
MediaControl.WindowFlags
```

See also

MediaControl (Page 3600)

MediaControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the Media Player.

Syntax

```
MediaControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

MediaControl (Page 3600)

MediaControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the Media Player.

Syntax

```
MediaControl.FireCommand(commandId, custom)
```

Parameters

`commandId`

Type: Int32

ID of the element of the toolbar or information bar

`custom`

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

MediaControl (Page 3600)

MediaControl.Pause()

Description

The "Pause" method pauses the playback.

Syntax

```
MediaControl.Pause()
```

Parameters

--

Return value

--

See also

MediaControl (Page 3600)

MediaControl.Play()

Description

The "Play" method starts or resumes playback.

Syntax

```
MediaControl.Play()
```

Parameters

--

Return value

See also

MediaControl (Page 3600)

MediaControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
MediaControl.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

MediaControl (Page 3600)

MediaControl.Stop()

Description

The "Stop" method stops playback.

Syntax

```
MediaControl.Stop()
```

Parameters

--

Return value

--

See also

MediaControl (Page 3600)

MediaControl_OnActivated()

Description

The "OnActivated" event occurs when a Media Player receives focus:

- A Media Player is selected via the configured tab sequence.
- A Media Player that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
MediaControl_OnActivated(item)
```

Context

item

Type: Object

Media Player where the event occurs.

See also

MediaControl (Page 3600)

MediaControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the Media Player.

Syntax

```
MediaControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Media Player where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

MediaControl (Page 3600)

MediaControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when a Media Player loses focus because the operator presses the <TAB> key or executes a different action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
MediaControl_OnDeactivated(item)
```

Context

item

Type: Object

Media Player where the event occurs.

See also

MediaControl (Page 3600)

MediaControl_OnInitialized()

Description

The "OnInitialized" event occurs when a Media Player has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
MediaControl_OnInitialized(item)
```

Context

item

Type: Object

Media Player where the event occurs.

See also

MediaControl (Page 3600)

MediaControl_OnPaused()

Description

The "OnPaused" event occurs when playback is paused in the Media Player.

Syntax

```
MediaControl_OnPaused(item)
```

Context

item

Type: Object

Media Player where the event occurs.

See also

MediaControl (Page 3600)

MediaControl_OnPlaying()

Description

The "OnPlaying" event occurs when playback is started or resumed in the Media Player.

Syntax

```
MediaControl_OnPlaying(item)
```

Context

item

Type: Object

Media Player where the event occurs.

See also

MediaControl (Page 3600)

MediaControl_OnStopped()

Description

The "OnStopped" event occurs when playback is stopped in the Media Player.

Syntax

```
MediaControl_OnStopped(item)
```

Context

item

Type: Object

Media Player where the event occurs.

See also

MediaControl (Page 3600)

ObjectExplorerControl

Description

The "ObjectExplorerControl" object specifies the properties of the display for building objects.

Object type

HmiObjectExplorerControl

Properties

The "ObjectExplorerControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the display.
- **DisplayViewType**
  Specifies the display type.

- **Enabled**
  Specifies whether the display can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the display.
- **Layer**
  Returns the screen layer in which the display is located.
- **Left**
  Specifies the value of the X coordinate.
- **LinkedItem**
  Specifies the referenced object.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the display.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the display was created.
- **ShowFocusVisual**
  Specifies whether the display is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the display.
- **StyleItemClass**
  Returns the style which is applied to the display.
- **TabIndex**
  Returns the position of the display in the tab sequence.
- **TargetObjectType**
  Specifies the object that will be displayed.
- **ToolBar**
  Specifies the toolbar of the display.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the display is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the display.

Methods

The "ObjectExplorerControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the display.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

ObjectExplorerControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ObjectExplorerControl.BackColor
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ObjectExplorerControl.Caption
```

See also

ObjectExplorerControl (Page 3789)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ObjectExplorerControl.Caption (Page 3791)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 3792)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 3792)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 3792)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 3792)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 3792)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 3792)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ObjectExplorerControl.Caption (Page 3791)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ObjectExplorerControl.Caption (Page 3791)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ObjectExplorerControl.Caption (Page 3791)

ObjectExplorerControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ObjectExplorerControl.CaptionColor
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the control.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ObjectExplorerControl.CurrentQuality
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.DisplayViewType

Description

The "DisplayViewType" property specifies the display type.

Type

Int32, HmiDisplayViewType

Specifies the display type.

- Standard (0): Default setting; usually list display
- List (1): List display without details
- Details (2): List display with details

- SmallIcons (3): Small icons
- MediumIcons (4): Medium icons
- LargeIcons (5): Large icons

Access

Read-write

Syntax

```
ObjectExplorerControl.DisplayViewType
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Enabled

Description

The "Enabled" property specifies whether the display can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ObjectExplorerControl.Enabled
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ObjectExplorerControl.Height
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Icon

Description

The "Icon" property specifies the icon of the display.

Type

String

Access

Read-write

Syntax

```
ObjectExplorerControl.Icon
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Layer

Description

The "Layer" property returns the screen layer in which the control is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ObjectExplorerControl.Layer
```

See also

ObjectExplorerControl (Page 3789)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ObjectExplorerControl.Layer (Page 3800)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ObjectExplorerControl.Layer (Page 3800)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ObjectExplorerControl.Layer (Page 3800)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ObjectExplorerControl.Layer (Page 3800)

ObjectExplorerControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ObjectExplorerControl.Left
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.LinkedItem

Description

The "LinkedItem" property specifies the referenced object.

Type

Object, HmiControlWindowBase (Page 1571)

Access

Read-write

Syntax

```
ObjectExplorerControl.LinkedItem
```

See also

ObjectExplorerControl (Page 3789)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ObjectExplorerControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ObjectExplorerControl.Margin
```

See also

ObjectExplorerControl (Page 3789)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ObjectExplorerControl.Margin (Page 3804)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ObjectExplorerControl.Margin (Page 3804)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ObjectExplorerControl.Margin (Page 3804)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ObjectExplorerControl.Margin (Page 3804)

ObjectExplorerControl.Name

Description

The "Name" property returns the name of the control.

Type

String

Access

Read-only

Syntax

```
ObjectExplorerControl.Name
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ObjectExplorerControl.Parent
```

See also

ObjectExplorerControl (Page 3789)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ObjectExplorerControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the control was created.

Type

String

Access

Read-only

Syntax

```
ObjectExplorerControl.RenderingTemplate
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the display is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ObjectExplorerControl.ShowFocusVisual
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the display.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
ObjectExplorerControl.StatusBar
```

See also

ObjectExplorerControl (Page 3789)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 3810)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

HmiControlBarElementCollection (Page 3810)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 3810)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 3810)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 3828)

See also

HmiControlBarElementCollection (Page 3810)

Control Bar Elements (Page 3828)

Control Bar Elements

Description

Control Bar Elements (Page 3828)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 3813)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 3813)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 3813)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 3813)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 3813)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 3813)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 3817)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 3817)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 3817)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 3817)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 3819)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 3819)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 3819)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 3819)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

ObjectExplorerControl.StatusBar (Page 3809)

ObjectExplorerControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the display.

Type

String

Access

Read-only

Syntax

```
ObjectExplorerControl.StyleItemClass
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.TabIndex

Description

The "TabIndex" property returns the position of the display in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ObjectExplorerControl.TabIndex
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.TargetObjectType

Description

The "TargetObjectType" property specifies the object that will be displayed.

Type

UInt32

Access

Read-write

Syntax

```
ObjectExplorerControl.TargetObjectType
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the display.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
ObjectExplorerControl.ToolBar
```

See also

ObjectExplorerControl (Page 3789)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 3826)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

HmiControlBarElementCollection (Page 3826)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 3825)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 3826)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 3828)

See also

HmiControlBarElementCollection (Page 3826)

Control Bar Elements (Page 3828)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.

- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 3828)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 3828)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 3828)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 3828)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 3828)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 3833)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 3833)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 3833)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 3833)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 3833)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above

- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 3833)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 3833)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 3833)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 3828)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 3828)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog

- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled

- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 3828)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 3843)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 3843)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 3843)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 3843)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 3828)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 3828)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 3828)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 3828)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 3849)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 3849)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 3849)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 3849)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 3828)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 3828)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 3828)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 3854)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 3856)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 3856)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 3856)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 3856)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 3856)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 3856)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 3856)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 3856)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 3854)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 3866)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 3866)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 3866)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 3866)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 3854)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 3871)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 3871)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 3871)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 3871)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 3854)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 3854)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 3876)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 3883)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 3883)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 3883)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 3883)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 3876)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 3888)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 3888)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 3888)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 3888)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 3876)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 3876)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 3893)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 3899)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 3899)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 3899)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 3899)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 3893)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 3904)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 3904)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 3904)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 3904)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 3893)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 3893)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 3909)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 3918)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 3918)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 3918)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 3918)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 3909)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 3923)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 3923)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 3923)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 3923)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 3909)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 3909)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 3929)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 3936)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 3936)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 3929)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 3947)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 3947)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 3947)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 3947)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 3929)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 3952)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 3952)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 3952)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 3952)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 3929)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 3929)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 3958)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 3958)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 3958)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 3958)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 3958)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 3958)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 3961)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 3961)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 3961)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 3961)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 3964)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 3964)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 3964)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 3964)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

ObjectExplorerControl.ToolBar (Page 3824)

ObjectExplorerControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ObjectExplorerControl.Top
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ObjectExplorerControl.Visible
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ObjectExplorerControl.Width
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the display.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title

- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
ObjectExplorerControl.WindowFlags
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the display.

Syntax

```
ObjectExplorerControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the display.

Syntax

```
ObjectExplorerControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

ObjectExplorerControl (Page 3789)

ObjectExplorerControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ObjectExplorerControl.PropertyFlashing(propertyName, enable[, value]
[, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ObjectExplorerControl (Page 3789)

OpenLinkElement

Description

The "OpenLinkElement" object displays the connection to a database.

Object type

HmiOpenLinkElement

Properties

The "OpenLinkElement" object has the following properties:

- **ActiveOnStartup**
  Returns whether the connection is active on startup.
- **Application**
  Specifies the application.
- **ContainedType**
  Returns the type of the contained properties.
- **OpenLinkMode**
  Returns the connection.
- **Parent**
  Returns the higher-level screen object.
- **Pipe**
  Specifies the data stream.
- **Properties**
  Enables access to the properties of the connected object.

Methods

The "OpenLinkElement" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the connection.
- **PropertyFlashing()**
  Configures flashing of a property.

- **Start()**
  Starts the connection.
- **Stop()**
  Terminates the connection.

OpenLinkElement.ActiveOnStartup

Description

The "ActiveOnStartup" property returns whether the connection is active on startup.

Type

Bool

Access

Read-only

Syntax

```
OpenLinkElement.ActiveOnStartup
```

See also

OpenLinkElement (Page 3973)

OpenLinkElement.Application

Description

The "Application" property specifies the application.

Type

Object, HmiApplicationPart

Access

Read-write

Syntax

```
OpenLinkElement.Application
```

See also

OpenLinkElement (Page 3973)

Application.ApplicationExitCode

Description

The "ApplicationExitCode" property returns the exit code of the application.

Type

UInt32

Access

Read-only

Syntax

```
Application.ApplicationExitCode
```

See also

OpenLinkElement.Application (Page 3974)

Application.ApplicationName

Description

The "ApplicationName" property returns the name of the application.

Type

String

Access

Read-only

Syntax

```
Application.ApplicationName
```

See also

OpenLinkElement.Application (Page 3974)

Application.ApplicationState

Description

The "ApplicationState" property returns the status of the application.

Type

Int32, HmiApplicationState

Returns the status.

- None (0): None
- Running (1): Running
- Terminated (2): Completed
- Crashed (3): Crashed
- CreateFailed (4): Create failed

Access

Read-only

Syntax

```
Application.ApplicationState
```

See also

OpenLinkElement.Application (Page 3974)

Application.Arguments

Description

The "Arguments" property returns the parameters of the application.

Type

String

Access

Read-only

Syntax

```
Application.Arguments
```

See also

OpenLinkElement.Application (Page 3974)

Application.Environment

Description

The "Environment" property returns the environment of the application.

Type

String

Access

Read-only

Syntax

```
Application.Environment
```

See also

OpenLinkElement.Application (Page 3974)

Application.TerminateApplicationOnLeave

Description

The "TerminateApplicationOnLeave" property returns whether the application is terminated on exit.

Type

Bool

Access

Read-only

Syntax

```
Application.TerminateApplicationOnLeave
```

See also

OpenLinkElement.Application (Page 3974)

Application.WorkingDirectory

Description

The "WorkingDirectory" property returns the working directory of the application.

Type

String

Access

Read-only

Syntax

```
Application.WorkingDirectory
```

See also

OpenLinkElement.Application (Page 3974)

OpenLinkElement.ContainedType

Description

The "ContainedType" property returns the type of the contained properties (CustomControl, SwacComponent, or WidgetType).

Type

String

Access

Read-only

Syntax

```
OpenLinkElement.ContainedType
```

See also

OpenLinkElement (Page 3973)

OpenLinkElement.OpenLinkMode

Description

The "OpenLinkMode" property returns the type of connection.

Type

Int32, HmiOpenLinkMode

Returns the type of connection.

- None (0): None
- Create (1): Create or end external process.
- CreateAndConnect (2): Create data stream and end data stream.
- Connect (3): Use a named data stream that already exists (connect and disconnect).

Access

Read-only

Syntax

```
OpenLinkElement.OpenLinkMode
```

See also

OpenLinkElement (Page 3973)

OpenLinkElement.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
OpenLinkElement.Parent
```

See also

OpenLinkElement (Page 3973)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

OpenLinkElement.Pipe

Description

The "Pipe" property specifies the data stream.

Type

Object, HmiPipePart

Access

Read-write

Syntax

```
OpenLinkElement.Pipe
```

See also

OpenLinkElement (Page 3973)

Pipe.CharSet

Description

The "CharSet" property specifies the character set of the data stream.

Type

Int32, HmiCharSet

Specifies the character set:

- Console (0): Depending on the setting of the operating system during runtime
- UTF8 (1): UTF-8
- UTF16 (2): UTF 16

Access

Read-write

Syntax

```
Pipe.CharSet
```

See also

OpenLinkElement.Pipe (Page 3980)

Pipe.LastPropertyRequestResult

Description

The "LastPropertyRequestResult" property returns the result of the last read or write operation of a property of the data stream.

Type

Int32, HmiRequestResult

Returns the result:

- None (0): None
- Success (1): Success
- AccessDenied (2): Access denied
- InvalidName (3): Invalid name

Access

Read-only

Syntax

```
Pipe.LastPropertyRequestResult
```

See also

OpenLinkElement.Pipe (Page 3980)

Pipe.PipeName

Description

The "PipeName" property returns the name of the data stream.

Type

String

Access

Read-only

Syntax

```
Pipe.PipeName
```

See also

OpenLinkElement.Pipe (Page 3980)

Pipe.PipeState

Description

The "PipeState" property returns the connection status of the data stream.

Type

Int32, HmiPipeConnectionState

Returns the connection status:

- Unavailable (0): Initial state or closed
- Connected (1): Connected
- Disconnected (2): Disconnected

Access

Read-only

Syntax

```
Pipe.PipeState
```

See also

OpenLinkElement.Pipe (Page 3980)

Pipe.ReconnectAutomatically

Description

The "ReconnectAutomatically" property returns whether the data stream is automatically reconnected.

Type

Bool

Access

Read-only

Syntax

```
Pipe.ReconnectAutomatically
```

See also

OpenLinkElement.Pipe (Page 3980)

OpenLinkElement.Properties

Description

The "Properties" property enables access to dynamic properties of the connected object (e.g. interface tags and interface properties of a faceplate type).

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
OpenLinkElement.Properties
```

See also

OpenLinkElement (Page 3973)

OpenLinkElement.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the connection.

Syntax

```
OpenLinkElement.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

OpenLinkElement (Page 3973)

OpenLinkElement.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
OpenLinkElement.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

OpenLinkElement (Page 3973)

OpenLinkElement.Start()

Description

The "Start" method starts the connection.

Syntax

```
OpenLinkElement.Start()
```

Parameters

--

Return value

--

See also

OpenLinkElement (Page 3973)

OpenLinkElement.Stop()

Description

The "Stop" method terminates the connection.

Syntax

```
OpenLinkElement.Stop()
```

Parameters

--

Return value

--

See also

OpenLinkElement (Page 3973)

OverviewParameterControl

Description

The "OverviewParameterControl" object represents a parameter overview in runtime.

Object type

HmiOverviewParameterControl

Properties

The "OverviewParameterControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the parameter overview.
- **DetailedParameterControl**
  Specifies the parameter set control of tag values.
- **EditMode**
  Specifies the editing mode for values in runtime.
- **Enabled**
  Specifies whether the parameter overview can be operated in runtime.
- **Filter**
  Specifies a string for filtering parameters.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the parameter overview.
- **Layer**
  Returns the screen layer in which the parameter overview is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the parameter overview.
- **ParameterSetTypeDefault**
  Specifies the standard parameter set type.

- **ParameterView**
  Defines the appearance of the parameter table.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the parameter overview was created.
- **ShowFocusVisual**
  Specifies whether the parameter overview is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the parameter overview.
- **StyleItemClass**
  Returns the style which is applied to the parameter overview.
- **TabIndex**
  Returns the position of the parameter overview in the tab sequence.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Specifies the toolbar of the parameter overview.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the parameter overview is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the parameter overview.

Methods

The "OverviewParameterControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the parameter overview.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

OverviewParameterControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControl.BackColor
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
OverviewParameterControl.Caption
```

See also

OverviewParameterControl (Page 3987)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

OverviewParameterControl.Caption (Page 3989)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 3989)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 3989)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 3989)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 3989)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 3989)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 3989)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

OverviewParameterControl.Caption (Page 3989)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

OverviewParameterControl.Caption (Page 3989)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

OverviewParameterControl.Caption (Page 3989)

OverviewParameterControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControl.CaptionColor
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the parameter overview.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
OverviewParameterControl.CurrentQuality
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.DetailedParameterControl

Description

The "DetailedParameterControl" property specifies the parameter set control of tag values.

Type

Object, HmiDetailedParameterControl (Page 2587)

Access

Read-write

Syntax

```
OverviewParameterControl.DetailedParameterControl
```

See also

OverviewParameterControl (Page 3987)

DetailedParameterControl (Page 2587)

DetailedParameterControl

Description

DetailedParameterControl (Page 2587)

OverviewParameterControl.EditMode

Description

The "EditMode" property specifies the editing mode for values in runtime.

Type

Int32, HmiEditMode

Specifies the editing mode:

- None (0): No access
- Update (1): Update values
- Create (2): Create values
- Delete (4): Delete values

Access

Read-write

Syntax

```
OverviewParameterControl.EditMode
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Enabled

Description

The "Enabled" property specifies whether the parameter overview can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
OverviewParameterControl.Enabled
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Filter

Description

The "Filter" property specifies a string for filtering parameters.

The syntax of the filter string corresponds to the WHERE clause of an SQL command.

Type

String

Access

Read-write

Syntax

```
OverviewParameterControl.Filter
```

Operators

The following operators can be used in the filter string:

Operator	Description	Example
`=`	equal to	`AlarmClassName = 'demo'`
`IS NOT string`	is not equal to the string *string*	`AlarmText4 IS NOT 'Text5'`
`<>`	not equal	`Value <> 0.0`
`>`	greater than	`ModificationTime > '11.08.2016'`
`<`	less than	`Value < 75.0`
`>=`	greater than or equal to	`Value >= 25.0`
`<=`	less than or equal to	`Value <= 75.0`
`OR, ||`	logical OR	`State = 1 OR State = 3`
`AND, &&`	logical AND	`EventText = 'Text1' AND Origin = 'Motor'`
`BETWEEN`	within a range	`Value BETWEEN 25.0 AND 75.0`
`NOT BETWEEN`	outside a range	`Value NOT BETWEEN 25.0 AND 75.0`
`LIKE string`	corresponds to the string *string*	`Name LIKE 'Motor*'`
`NOT LIKE string`	does not correspond to the string *string*	`Name NOT LIKE 'Valve*'`
`IN (v1, v2, …)`	corresponds to one or more values	`State IN (1, 4, 7)`
`NOT IN (v1, v2, …)`	does not correspond to one or more values	`State NOT IN (0, 2, 3, 5, 6)`
`IS NULL`	compares to zero (missing data)	`Context IS NULL`
`IS NOT NULL`	compares to zero (unknown data)	`Context IS NOT NULL`

Wildcards

The following wildcards can be used for characters of filter strings:

Wildcard	Description	Example
*	replaces 0, 1 or more characters	`Name LIKE 'Motor*'`
?	replaces exactly 1 character	`Name = 'Recipe?'`

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControl.Height
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Icon

Description

The "Icon" property specifies the icon of the parameter overview.

Type

String

Access

Read-write

Syntax

```
OverviewParameterControl.Icon
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Layer

Description

The "Layer" property returns the screen layer in which the parameter overview is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
OverviewParameterControl.Layer
```

See also

OverviewParameterControl (Page 3987)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

OverviewParameterControl.Layer (Page 4000)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

OverviewParameterControl.Layer (Page 4000)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

OverviewParameterControl.Layer (Page 4000)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

OverviewParameterControl.Layer (Page 4000)

OverviewParameterControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
OverviewParameterControl.Left
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
OverviewParameterControl.Margin
```

See also

OverviewParameterControl (Page 3987)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

OverviewParameterControl.Margin (Page 4003)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

OverviewParameterControl.Margin (Page 4003)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

OverviewParameterControl.Margin (Page 4003)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

OverviewParameterControl.Margin (Page 4003)

OverviewParameterControl.Name

Description

The "Name" property returns the name of the parameter overview.

Type

String

Access

Read-only

Syntax

```
OverviewParameterControl.Name
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.ParameterSetTypeDefault

Description

The "ParameterSetTypeDefault" property specifies the standard parameter set type.

Type

Object, HmiParameterSetType (Page 1234)

Access

Read-write

Syntax

```
OverviewParameterControl.ParameterSetTypeDefault
```

See also

OverviewParameterControl (Page 3987)

ParameterSetType (Page 1234)

ParameterSetType

Description

ParameterSetType (Page 1234)

OverviewParameterControl.ParameterView

Description

The "ParameterView" property defines the appearance of the parameter table.

Type

Object, HmiDataGridViewPart (Page 4007)

Access

Read-write

Syntax

```
OverviewParameterControl.ParameterView
```

See also

DataGridView (Page 4007)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

OverviewParameterControl.ParameterView (Page 4006)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 4009)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 4009)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 4009)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 4009)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 4013)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

OverviewParameterControl.ParameterView (Page 4006)

HmiDataGridColumnCollection (Page 4013)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

See also

DataGridView.Columns (Page 4013)

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 4013)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 4015)

See also

HmiDataGridColumnCollection (Page 4013)

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn

Description

The "OverviewParameterControlColumn" object represents a value column.

Object type

HmiOverviewParameterControlColumnPart

Properties

The "OverviewParameterControlColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies display options for text and graphics.

- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **Key**
  Corresponds to the column definition from the "ConsideredColumns" property of the connected source.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **OverviewParameterControlBlock**
  Specifies the information blocks.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection (Page 4013)

OverviewParameterControlColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
OverviewParameterControlColumn.AllowSort
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControlColumn.BackColor
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
OverviewParameterControlColumn.Content
```

See also

OverviewParameterControlColumn (Page 4015)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

OverviewParameterControlColumn.Content (Page 4017)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

OverviewParameterControlColumn.Content (Page 4017)

OverviewParameterControlColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
OverviewParameterControlColumn.Enabled
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControlColumn.ForeColor
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
OverviewParameterControlColumn.Header
```

See also

OverviewParameterControlColumn (Page 4015)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

OverviewParameterControlColumn.Header (Page 4024)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 4024)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4024)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

OverviewParameterControlColumn.Header (Page 4024)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

OverviewParameterControlColumn.Header (Page 4024)

OverviewParameterControlColumn.Key

Description

The "Key" property corresponds to the column definition from the "ConsideredColumns" property of the connected source.

Type

String

Access

Read-only

Syntax

```
OverviewParameterControlColumn.Key
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControlColumn.MaximumWidth
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControlColumn.MinimumWidth
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
OverviewParameterControlColumn.Name
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
OverviewParameterControlColumn.OutputFormat
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.OverviewParameterControlBlock

Description

The "OverviewParameterControlBlock" property specifies the information blocks.

Type

Int32, HmiOverviewParameterControlBlock

Specifies the information blocks:

- None (0): None
- ParameterSetID (1): ID
- LastUser (2): Last user
- LastAccess (3): Last access
- ParameterSetElementOdd (4): Generic columns (odd numbering)
- ParameterSetElementEven (5): Generic columns (even numbering)

Access

Read-write

Syntax

```
OverviewParameterControlColumn.OverviewParameterControlBlock
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
OverviewParameterControlColumn.SortDirection
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.SortOrder

Description

The "SortOrder" property specifies the order of the sorting.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
OverviewParameterControlColumn.SortOrder
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
OverviewParameterControlColumn.Visible
```

See also

OverviewParameterControlColumn (Page 4015)

OverviewParameterControlColumn.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControlColumn.Width
```

See also

OverviewParameterControlColumn (Page 4015)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

OverviewParameterControl.ParameterView (Page 4006)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 4036)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 4036)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 4036)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 4036)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 4036)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 4036)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 4042)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 4044)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 4044)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 4044)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 4044)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 4044)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 4044)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 4042)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 4042)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.SelectionBorderWidth

Description

The "SelectionBorderWidth" property specifies the border width of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderWidth
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

OverviewParameterControl.ParameterView (Page 4006)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

OverviewParameterControl.ParameterView (Page 4006)

OverviewParameterControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
OverviewParameterControl.Parent
```

See also

OverviewParameterControl (Page 3987)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

OverviewParameterControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the parameter overview was created.

Type

String

Access

Read-only

Syntax

```
OverviewParameterControl.RenderingTemplate
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the parameter overview is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
OverviewParameterControl.ShowFocusVisual
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the parameter overview.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
OverviewParameterControl.StatusBar
```

See also

OverviewParameterControl (Page 3987)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

OverviewParameterControl.StatusBar (Page 4057)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 4059)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

OverviewParameterControl.StatusBar (Page 4057)

HmiControlBarElementCollection (Page 4059)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 4058)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4059)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4076)

See also

HmiControlBarElementCollection (Page 4059)

Control Bar Elements (Page 4076)

Control Bar Elements

Description

Control Bar Elements (Page 4076)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

OverviewParameterControl.StatusBar (Page 4057)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

OverviewParameterControl.StatusBar (Page 4057)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 4061)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 4061)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 4061)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 4061)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 4061)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 4061)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

OverviewParameterControl.StatusBar (Page 4057)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 4065)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 4065)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 4065)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 4065)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

OverviewParameterControl.StatusBar (Page 4057)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 4068)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 4068)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 4068)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 4068)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

OverviewParameterControl.StatusBar (Page 4057)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

OverviewParameterControl.StatusBar (Page 4057)

OverviewParameterControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the parameter overview.

Type

String

Access

Read-only

Syntax

```
OverviewParameterControl.StyleItemClass
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.TabIndex

Description

The "TabIndex" property returns the position of the parameter overview in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
OverviewParameterControl.TabIndex
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.TimeZone

Description

The "TimeZone" property specifies the time zone.

Type

Int32, HmiTimeZone

Access

Read-write

Syntax

```
OverviewParameterControl.TimeZone
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the parameter overview.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
OverviewParameterControl.ToolBar
```

See also

OverviewParameterControl (Page 3987)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

OverviewParameterControl.ToolBar (Page 4073)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 4074)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

OverviewParameterControl.ToolBar (Page 4073)

HmiControlBarElementCollection (Page 4074)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 4074)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4074)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4076)

See also

HmiControlBarElementCollection (Page 4074)

Control Bar Elements (Page 4076)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 4076)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 4076)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 4076)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 4076)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 4076)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 4082)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 4082)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 4082)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 4082)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 4082)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 4082)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 4082)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 4082)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 4076)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 4076)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 4076)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 4092)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 4092)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 4092)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 4092)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 4076)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 4076)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 4076)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 4076)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 4097)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 4097)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 4097)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 4097)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 4076)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 4076)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 4076)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 4102)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 4104)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 4104)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4104)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 4104)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 4104)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 4104)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 4104)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4104)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 4102)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 4114)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 4114)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 4114)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 4114)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 4102)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 4119)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 4119)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 4119)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 4119)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 4102)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 4102)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.

- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms

- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export

- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 4124)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 4131)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 4131)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 4131)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 4131)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 4124)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 4136)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 4136)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 4136)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 4136)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 4124)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 4124)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 4142)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 4148)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 4148)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 4148)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 4148)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 4142)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 4153)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 4153)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 4153)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 4153)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 4142)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 4142)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.

- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 4158)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 4167)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 4167)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 4167)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 4167)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 4158)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 4172)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 4172)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 4172)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 4172)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 4158)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 4158)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.

- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 4178)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 4184)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 4184)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 4178)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 4195)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 4195)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 4195)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 4195)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 4178)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 4200)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 4200)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 4200)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 4200)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 4178)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 4178)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

OverviewParameterControl.ToolBar (Page 4073)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

OverviewParameterControl.ToolBar (Page 4073)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 4206)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 4206)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 4206)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 4206)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 4206)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 4206)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

OverviewParameterControl.ToolBar (Page 4073)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 4210)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 4210)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 4210)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 4210)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

OverviewParameterControl.ToolBar (Page 4073)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 4212)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 4212)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 4212)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 4212)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

OverviewParameterControl.ToolBar (Page 4073)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

OverviewParameterControl.ToolBar (Page 4073)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

OverviewParameterControl.ToolBar (Page 4073)

OverviewParameterControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
OverviewParameterControl.Top
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Visible

Description

The "Visible" property specifies whether the parameter overview is visible.

Type

Bool

Access

Read-write

Syntax

```
OverviewParameterControl.Visible
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
OverviewParameterControl.Width
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the parameter overview.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
OverviewParameterControl.WindowFlags
```

Example

Adapting the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the parameter overview.

Syntax

```
OverviewParameterControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the parameter overview.

Syntax

```
OverviewParameterControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

OverviewParameterControl (Page 3987)

OverviewParameterControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
OverviewParameterControl.PropertyFlashing(propertyName, enable[,
value][, alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

OverviewParameterControl (Page 3987)

Polygon

Description

The "Polygon" object represents a polygon in runtime.

Object type

HmiPolygon

Properties

The "Polygon" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the polygon.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the polygon can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the polygon is filled.
- **FillLevel**
  Specifies the fill of the polygon in percent.
- **Height**
  Specifies the height.
- **JoinType**
  Specifies the corner style of the polygon
- **Layer**
  Returns the layer of the screen in which the polygon is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the polygon.

- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the polygon is operable.
- **Parent**
  Returns the higher-level screen object.
- **Points**
  Specifies the coordinates of the polygon points.
- **RequireExplicitUnlock**
  Returns whether the polygon is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the polygon rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the polygon is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the polygon.
- **TabIndex**
  Returns the position of the polygon in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the polygon is visible.
- **Width**
  Specifies the width.

Methods

The "Polygon" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the polygon.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Polygon" object has the following events:

- **OnActivated()**
  Occurs when a polygon receives focus.
- **OnContextTapped()**
  Occurs when a polygon is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a polygon loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the polygon is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the polygon is in focus.
- **OnTapped()**
  Occurs when a polygon is left-clicked or short-touched.

Polygon.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Polygon.AlternateBackColor
```

See also

Polygon (Page 4221)

Polygon.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Polygon.AlternateBorderColor
```

See also

Polygon (Page 4221)

Polygon.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Polygon.Authorization
```

See also

Polygon (Page 4221)

Polygon.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Polygon.BackColor
```

See also

Polygon (Page 4221)

Polygon.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
Polygon.BackFillPattern
```

See also

Polygon (Page 4221)

Polygon.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Polygon.BorderColor
```

See also

Polygon (Page 4221)

Polygon.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Polygon.BorderWidth
```

See also

Polygon (Page 4221)

Polygon.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the polygon.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Polygon.CurrentQuality
```

See also

Polygon (Page 4221)

Polygon.DashType

Description

The "DashType" property specifies the stroke type of the border or line.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Polygon.DashType
```

See also

Polygon (Page 4221)

Polygon.Enabled

Description

The "Enabled" property specifies whether the polygon can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Polygon.Enabled
```

See also

Polygon (Page 4221)

Polygon.FillDirection

Description

The "FillDirection" property specifies the direction from which the polygon is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom
- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
Polygon.FillDirection
```

See also

Polygon (Page 4221)

Polygon.FillLevel

Description

The "FillLevel" property specifies the fill level of the polygons in percent.

Type

UInt8

Access

Read-write

Syntax

```
Polygon.FillLevel
```

See also

Polygon (Page 4221)

Polygon.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Polygon.Height
```

See also

Polygon (Page 4221)

Polygon.JoinType

Description

The "JoinType" property specifies the corner style of the polygon.

Type

Int32, HmiLineJoinType

Specifies the corner style:

- Round (0): Round
- Bevel (4096): Smooth
- Miter (8192): Miter

Access

Read-write

Syntax

```
Polygon.JoinType
```

See also

Polygon (Page 4221)

Polygon.Layer

Description

The "Layer" property returns the layer of the screen in which the polygon is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Polygon.Layer
```

See also

Polygon (Page 4221)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Polygon.Layer (Page 4232)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Polygon.Layer (Page 4232)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Polygon.Layer (Page 4232)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Polygon.Layer (Page 4232)

Polygon.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Polygon.Left
```

See also

Polygon (Page 4221)

Polygon.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Polygon.Margin
```

See also

Polygon (Page 4221)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Polygon.Margin (Page 4235)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Polygon.Margin (Page 4235)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Polygon.Margin (Page 4235)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Polygon.Margin (Page 4235)

Polygon.Name

Description

The "Name" property returns the name of the polygon.

Type

String

Access

Read-only

Syntax

```
Polygon.Name
```

See also

Polygon (Page 4221)

Polygon.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Polygon.Opacity
```

See also

Polygon (Page 4221)

Polygon.Operability

Description

The "Operability" property returns whether the polygon is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Polygon.Operability
```

See also

Polygon (Page 4221)

Polygon.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Polygon.Parent
```

See also

Polygon (Page 4221)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Polygon.Points

Description

The "Points" property specifies the coordinates of the polygon points.

Type

Variant

Access

Read-write

Syntax

```
Polygon.Points
```

Example

Output the coordinates of the polygon "Polygon_1" with 3 points via debug output in runtime:

Copy code

```
const p = Screen.Items('Polygon_1').Points;
HMIRuntime.Trace("x1=" + p[0] + ", y1=" + p[1] + ", x2=" + p[2] + ", y2=" + p[3] + ", x3="
+ p[4] + ", y3=" + p[5]);
```

See also

Polygon (Page 4221)

Polygon.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the polygon can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Polygon.RequireExplicitUnlock
```

See also

Polygon (Page 4221)

Polygon.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Polygon.RotationAngle
```

See also

Polygon (Page 4221)

Polygon.RotationCenterPlacement (Page 4241)

Polygon.RotationCenterX (Page 4242)

Polygon.RotationCenterY (Page 4243)

Polygon.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the polygon rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Polygon.RotationCenterPlacement
```

See also

Polygon (Page 4221)

Polygon.RotationAngle (Page 4241)

Polygon.RotationCenterX (Page 4242)

Polygon.RotationCenterY (Page 4243)

Polygon.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Polygon.RotationCenterX
```

See also

Polygon (Page 4221)

Polygon.RotationAngle (Page 4241)

Polygon.RotationCenterPlacement (Page 4241)

Polygon.RotationCenterY (Page 4243)

Polygon.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Polygon.RotationCenterY
```

See also

Polygon (Page 4221)

Polygon.RotationAngle (Page 4241)

Polygon.RotationCenterPlacement (Page 4241)

Polygon.RotationCenterX (Page 4242)

Polygon.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
Polygon.ShowFillLevel
```

See also

Polygon (Page 4221)

Polygon.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the polygon is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Polygon.ShowFocusVisual
```

See also

Polygon (Page 4221)

Polygon.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the polygon.

Type

String

Access

Read-only

Syntax

```
Polygon.StyleItemClass
```

See also

Polygon (Page 4221)

Polygon.TabIndex

Description

The "TabIndex" property returns the position of the polygon in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Polygon.TabIndex
```

See also

Polygon (Page 4221)

Polygon.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Polygon.ToolTipText
```

See also

Polygon (Page 4221)

Polygon.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Polygon.Top
```

See also

Polygon (Page 4221)

Polygon.Visible

Description

The "Visible" property specifies whether the polygon is visible.

Type

Bool

Access

Read-write

Syntax

```
Polygon.Visible
```

See also

Polygon (Page 4221)

Polygon.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Polygon.Width
```

See also

Polygon (Page 4221)

Polygon.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the polygon.

Syntax

```
Polygon.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Polygon (Page 4221)

Polygon.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Polygon.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Polygon (Page 4221)

Polygon_OnActivated()

Description

The "OnActivated" event occurs when a polygon receives focus:

- A polygon is selected via the configured tab sequence.
- A polygon that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Polygon_OnActivated(item)
```

Context

item

Type: Object

Polygon where the event occurs.

See also

Polygon (Page 4221)

Polygon_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A polygon is right-clicked.
- A polygon is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Polygon_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Polygon where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Polygon (Page 4221)

Polygon_OnDeactivated()

Description

The "OnDeactivated" event occurs when the polygon loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Polygon_OnDeactivated(item)
```

Context

`item`

Type: Object

Polygon where the event occurs.

See also

Polygon (Page 4221)

Polygon_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the polygon is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Polygon_OnKeyDown(item, keyCode, modifiers)
```

Context

`item`

Type: Object

Polygon where the event occurs.

`keyCode`

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Polygon (Page 4221)

Polygon_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the polygon is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Polygon_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Polygon where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Polygon (Page 4221)

Polygon_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A polygon is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a polygon has the focus.
- A polygon is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Polygon_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Polygon where the event occurs.

x

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Polygon (Page 4221)

Polyline

Description

The "Polyline" object represents a polyline in runtime.

Object type

HmiPolyline

Properties

The "Polyline" object has the following properties:

- **AlternateLineColor**
  Specifies the second line color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **CapType**
  Specifies the shape of the line ends.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the polyline.
- **DashType**
  Specifies the stroke style of the polyline.
- **Enabled**
  Specifies whether the polyline can be operated in runtime.
- **EndType**
  Specifies the type of line end.
- **Height**
  Specifies the height.
- **JoinType**
  Specifies the corner style of the polyline.
- **Layer**
  Returns the layer of the screen in which the polyline is located.
- **Left**
  Specifies the value of the X coordinate.
- **LineColor**
  Specifies the line color.
- **LineWidth**
  Specifies the line thickness.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the polyline.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the polyline is operable.
- **Parent**
  Returns the higher-level screen object.

- **Points**
  Specifies the coordinates of the polyline points.
- **RequireExplicitUnlock**
  Returns whether the polyline is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the polyline rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the polyline is highlighted when in focus.
- **StartType**
  Specifies the type of line start.
- **StyleItemClass**
  Returns the style which is applied to the polyline.
- **TabIndex**
  Returns the position of the polyline in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the polyline is visible.
- **Width**
  Specifies the width.

Methods

The "Polyline" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the polyline.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Polyline" object has the following events:

- **OnActivated()**
  Occurs when a polyline receives focus.
- **OnContextTapped()**
  Occurs when a polyline is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a polyline loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the polyline is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the polyline is in focus.
- **OnTapped()**
  Occurs when a polyline is left-clicked or short-touched.

Polyline.AlternateLineColor

Description

The "AlternateLineColor" property specifies the second line color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Polyline.AlternateLineColor
```

See also

Polyline (Page 4255)

Polyline.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Polyline.Authorization
```

See also

Polyline (Page 4255)

Polyline.CapType

Description

The "CapType" property specifies the shape of the line ends.

Type

Int32, HmiCapType

Specifies the line ends:

- Round (0): Round (line extends beyond the line end point with half the line thickness)
- Square (256): Square (line extends beyond the line end point with half the line thickness)
- Flat (512): Justified (line ends at the line end point)

Access

Read-write

Syntax

```
Polyline.CapType
```

See also

Polyline (Page 4255)

Polyline.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the polyline.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Polyline.CurrentQuality
```

See also

Polyline (Page 4255)

Polyline.DashType

Description

The "DashType" property specifies the stroke style of the polyline.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted

- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Polyline.DashType
```

See also

Polyline (Page 4255)

Polyline.Enabled

Description

The "Enabled" property specifies whether the polyline can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Polyline.Enabled
```

See also

Polyline (Page 4255)

Polyline.EndType

Description

The "EndType" property specifies the line end type.

Type

Int32, HmiLineEndType

Specifies the line end type:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reverse arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
Polyline.EndType
```

See also

Polyline (Page 4255)

Polyline.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Polyline.Height
```

See also

Polyline (Page 4255)

Polyline.JoinType

Description

The "JoinType" property specifies the corner style of the polyline.

Type

Int32, HmiLineJoinType

Specifies the corner style:

- Round (0): Round
- Bevel (4096): Smooth
- Miter (8192): Miter

Access

Read-write

Syntax

```
Polyline.JoinType
```

See also

Polyline (Page 4255)

Polyline.Layer

Description

The "Layer" property returns the layer of the screen in which the polyline is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Polyline.Layer
```

See also

Polyline (Page 4255)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Polyline.Layer (Page 4263)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Polyline.Layer (Page 4263)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Polyline.Layer (Page 4263)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Polyline.Layer (Page 4263)

Polyline.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Polyline.Left
```

See also

Polyline (Page 4255)

Polyline.LineColor

Description

The "LineColor" property specifies the line color.

Type

UInt32

Access

Read-write

Syntax

```
Polyline.LineColor
```

See also

Polyline (Page 4255)

Polyline.LineWidth

Description

The "LineWidth" property specifies the line thickness.

Type

UInt8

Access

Read-write

Syntax

```
Polyline.LineWidth
```

See also

Polyline (Page 4255)

Polyline.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Polyline.Margin
```

See also

Polyline (Page 4255)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Polyline.Margin (Page 4267)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Polyline.Margin (Page 4267)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Polyline.Margin (Page 4267)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Polyline.Margin (Page 4267)

Polyline.Name

Description

The "Name" property returns the name of the polyline.

Type

String

Access

Read-only

Syntax

```
Polyline.Name
```

See also

Polyline (Page 4255)

Polyline.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Polyline.Opacity
```

See also

Polyline (Page 4255)

Polyline.Operability

Description

The property "Operability" returns whether the polyline is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Polyline.Operability
```

See also

Polyline (Page 4255)

Polyline.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Polyline.Parent
```

See also

Polyline (Page 4255)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Polyline.Points

Description

The "Points" polyline specifies the coordinates of the points of the polyline.

Type

Variant

Access

Read-write

Syntax

```
Polyline.Points
```

Example

Output the coordinates of the polyline "Polyline_1" with 3 points via debug output in runtime:

Copy code

```
const p = Screen.Items('Polyline_1').Points;
HMIRuntime.Trace("x1=" + p[0] + ", y1=" + p[1] + ", x2=" + p[2] + ", y2=" + p[3] + ", x3="
+ p[4] + ", y3=" + p[5]);
```

See also

Polyline (Page 4255)

Polyline.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the polyline can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Polyline.RequireExplicitUnlock
```

See also

Polyline (Page 4255)

Polyline.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Polyline.RotationAngle
```

See also

Polyline (Page 4255)

Polyline.RotationCenterPlacement (Page 4274)

Polyline.RotationCenterX (Page 4274)

Polyline.RotationCenterY (Page 4275)

Polyline.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the polyline rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Polyline.RotationCenterPlacement
```

See also

Polyline (Page 4255)

Polyline.RotationAngle (Page 4273)

Polyline.RotationCenterX (Page 4274)

Polyline.RotationCenterY (Page 4275)

Polyline.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Polyline.RotationCenterX
```

See also

Polyline (Page 4255)

Polyline.RotationAngle (Page 4273)

Polyline.RotationCenterPlacement (Page 4274)

Polyline.RotationCenterY (Page 4275)

Polyline.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Polyline.RotationCenterY
```

See also

Polyline (Page 4255)

Polyline.RotationAngle (Page 4273)

Polyline.RotationCenterPlacement (Page 4274)

Polyline.RotationCenterX (Page 4274)

Polyline.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the polyline is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Polyline.ShowFocusVisual
```

See also

Polyline (Page 4255)

Polyline.StartType

Description

The "StartType" property specifies the type of line start.

Type

Int32, HmiLineEndType

Specifies the start of the line:

- Line (0): Line
- EmptyArrow (1): Empty arrow
- Arrow (2): Arrow
- ReversedArrow (3): Reverse arrow
- EmptyCircle (5): Empty circle
- Circle (6): Circle

Access

Read-write

Syntax

```
Polyline.StartType
```

See also

Polyline (Page 4255)

Polyline.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the polyline.

Type

String

Access

Read-only

Syntax

```
Polyline.StyleItemClass
```

See also

Polyline (Page 4255)

Polyline.TabIndex

Description

The "TabIndex" property returns the position of the polyline in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Polyline.TabIndex
```

See also

Polyline (Page 4255)

Polyline.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Polyline.ToolTipText
```

See also

Polyline (Page 4255)

Polyline.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Polyline.Top
```

See also

Polyline (Page 4255)

Polyline.Visible

Description

The "Visible" property specifies whether the polyline is visible.

Type

Bool

Access

Read-write

Syntax

```
Polyline.Visible
```

See also

Polyline (Page 4255)

Polyline.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Polyline.Width
```

See also

Polyline (Page 4255)

Polyline.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the polyline.

Syntax

```
Polyline.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Polyline (Page 4255)

Polyline.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Polyline.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Polyline (Page 4255)

Polyline_OnActivated()

Description

The "OnActivated" event occurs when a polyline receives focus:

- A polyline is selected via the configured tab sequence.
- A polyline that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Polyline_OnActivated(item)
```

Context

`item`

Type: Object

Polyline where the event occurs.

See also

Polyline (Page 4255)

Polyline_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A polyline is right-clicked.
- A polyline is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Polyline_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Polyline where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Polyline (Page 4255)

Polyline_OnDeactivated()

Description

The "OnDeactivated" event occurs when a polyline loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Polyline_OnDeactivated(item)
```

Context

`item`

Type: Object

Polyline where the event occurs.

See also

Polyline (Page 4255)

Polyline_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the polyline is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Polyline_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Polyline where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Polyline (Page 4255)

Polyline_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the polyline is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Polyline_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Polyline where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Polyline (Page 4255)

Polyline_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A polyline is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a polyline has the focus.
- A polyline is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Polyline_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Polyline where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Polyline (Page 4255)

PopupScreenWindow

Description

The "PopupScreenWindow" object represents a popup screen window.

Object type

HMIPopupScreenWindow

Properties

The "PopupScreenWindow" object has the following properties:

- **Adaption**
  Specifies how the window size adapts.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the popup screen window.
- **CurrentScreen**
  Returns the screen of the current popup screen window.
- **CurrentZoomFactor**
  Specifies the zoom factor which is applied to the displayed screen.
- **Enabled**
  Specifies whether the popup screen window can be operated in runtime.
- **Height**
  Specifies the height.
- **HorizontalScrollBarPosition**
  Specifies the horizontal alignment for the scroll bar.
- **HorizontalScrollBarVisibility**
  Specifies the setting for the horizontal scroll bar of the window.
- **Icon**
  Specifies the icon.
- **IsModal**
  Returns whether the popup screen window is modal.
- **InteractiveZooming**
  Specifies whether zooming is supported.
- **Layer**
  Returns the layer of the screen where the popup screen window is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin of the popup screen window.
- **Monitor**
  Returns the monitor on which the popup screen window is displayed.
- **Name**
  Returns the name of the popup screen window .
- **Parent**
  Returns the parent screen object (Parent container).
- **Path**
  Returns the absolute object path of the popup screen window.

- **RenderingTemplate**
  Returns the name of the template from which the object was created.
- **Screen**
  Specifies the name of the screen contained in the pop-up screen window.
- **ScreenName**
  Returns the screen name.
- **ScreenNumber**
  Returns the screen number.
- **ShowFocusVisual**
  Specifies whether the popup screen window is highlighted when in focus.
- **StartupPosition**
  Specifies the position of the popup screen window at runtime start.
- **StyleItemClass**
  Returns the style applied to the popup screen window.
- **System**
  Specifies the server prefix.
- **TabIndex**
  Returns the position of the screen window in the tab sequence.
- **TabIntoWindow**
  Specifies that the configured tab sequence of the displayed screen is resumed on activation via the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **VerticalScrollBarPosition**
  Specifies the vertical alignment for the scroll bar.
- **VerticalScrollBarVisibility**
  Specifies the setting for the vertical scroll bar of the window.
- **Visible**
  Specifies whether the popup screen window is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the screen window configuration.

Methods

The "PopupScreenWindow" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the screen window.
- **Close()**
  Closes the screen window.
- **PropertyFlashing()**
  Configures flashing of a property.

PopupScreenWindow.Adaption

Description

The "Adaption" property specifies how the window size adapts.

Type

Int32, HmiScreenWindowAdaption

Specifies how the window size adapts:

- None (0): No adaptation
- WindowToScreen (1): Window size corresponds to screen size
- ScreenToWindow (2): Screen is scaled to window size.

Access

Read-write

Syntax

```
PopupScreenWindow.Adaption
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
PopupScreenWindow.Caption
```

See also

PopupScreenWindow (Page 4288)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

PopupScreenWindow.Caption (Page 4291)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 4292)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 4292)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 4292)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 4292)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 4292)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 4292)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

PopupScreenWindow.Caption (Page 4291)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

PopupScreenWindow.Caption (Page 4291)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

PopupScreenWindow.Caption (Page 4291)

PopupScreenWindow.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
PopupScreenWindow.CaptionColor
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.CurrentQuality

Description

The "CurrentQuality" property returns the current worst quality code of all tags which influence the specified popup screen window.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable. Quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
PopupScreenWindow.CurrentQuality
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.CurrentScreen

Description

The "CurrentScreen" property returns the screen of the current popup screen window.

Type

Object, HmiScreen (Page 1397)

Access

Read-only

Syntax

```
PopupScreenWindow.CurrentScreen
```

See also

PopupScreenWindow (Page 4288)

Screen (Page 1397)

Screen

Description

Screen (Page 1397)

PopupScreenWindow.CurrentZoomFactor

Description

The "CurrentZoomFactor" property specifies the zoom factor of the popup screen window. The zoom factor may differ from the containing screen. The value 1.0 corresponds to a zoom factor of 100%.

Type

Float

Access

Read-write

Syntax

```
PopupScreenWindow.CurrentZoomFactor
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Enabled

Description

The "Enabled" property specifies whether the popup screen window can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
PopupScreenWindow.Enabled
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Height

Description

The "Height" property specifies the height of the popup screen window in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
PopupScreenWindow.Height
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.HorizontalScrollBarPosition

Description

The "HorizontalScrollBarPosition" property specifies the horizontal alignment for the scroll bar.

Type

Int32

Access

Read-write

Syntax

```
PopupScreenWindow.HorizontalScrollBarPosition
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
PopupScreenWindow.HorizontalScrollBarVisibility
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Icon

Description

The "Icon" property specifies the icon of the popup screen window.

Type

String

Access

Read-write

Syntax

```
PopupScreenWindow.Icon
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.InteractiveZooming

Description

The "InteractiveZooming" property specifies whether zooming is supported.

Type

Bool

Access

Read-write

Syntax

```
PopupScreenWindow.InteractiveZooming
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.IsModal

Description

The "IsModal" property returns whether the popup screen window is modal.

Type

Bool

Access

Read-only

Syntax

```
PopupScreenWindow.IsModal
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Layer

Description

The "Layer" property returns the layer of the screen where the popup screen window is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
PopupScreenWindow.Layer
```

See also

PopupScreenWindow (Page 4288)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

PopupScreenWindow.Layer (Page 4303)

Layer.MinimumZoom (Page 4304)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

PopupScreenWindow.Layer (Page 4303)

Layer.MaximumZoom (Page 4303)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

PopupScreenWindow.Layer (Page 4303)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

PopupScreenWindow.Layer (Page 4303)

PopupScreenWindow.Left

Description

The "Left" property specifies the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
PopupScreenWindow.Left
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Margin

Description

The "Margin" property specifies the surrounded outer distance of the popup screen window.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
PopupScreenWindow.Margin
```

See also

PopupScreenWindow (Page 4288)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

PopupScreenWindow.Margin (Page 4306)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

PopupScreenWindow.Margin (Page 4306)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

PopupScreenWindow.Margin (Page 4306)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

PopupScreenWindow.Margin (Page 4306)

PopupScreenWindow.Monitor

Description

The "Monitor" property returns the monitor on which the window is displayed.

Type

UInt8

Access

Read-only

Syntax

```
PopupScreenWindow.Monitor
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Name

Description

The "Name" property returns the name of the pop- up screen window.

Type

String

Access

Read-only

Syntax

```
PopupScreenWindow.Name
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Parent

Description

The "Parent" property returns the parent screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
PopupScreenWindow.Parent
```

See also

PopupScreenWindow (Page 4288)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

PopupScreenWindow.Path

Description

The "Path" property returns the absolute object path of the popup screen window in runtime, starting from the top-level screen window.

Note

For the syntax of an object path, see the "FindItem" method (Page 1549).

Type

String

Access

Read-only

Syntax

```
PopupScreenWindow.Path
```

See also

UI.FindItem() (Page 1549)

PopupScreenWindow (Page 4288)

PopupScreenWindow.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the object has been created.

Type

String

Access

Read-only

Syntax

```
PopupScreenWindow.RenderingTemplate
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Screen

Description

The "Screen" property specifies the name of the screen ("HMIScreen" type) contained in the pop-up screen window. Loads a new screen via its name into the popup screen window.

The "Screen" property returns a different value than the "CurrentScreen" when the referenced screen is not yet loaded completely or does not exist.

Type

String, HmiStoredScreen

Access

Read-write

Syntax

```
PopupScreenWindow.Screen
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.ScreenName

Description

The "ScreenName" property returns the screen name.

Type

String

Access

Read-only

Syntax

```
PopupScreenWindow.ScreenName
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.ScreenNumber

Description

The "ScreenNumber" property returns the screen number.

Type

UInt16

Access

Read-only

Syntax

```
PopupScreenWindow.ScreenNumber
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the popup screen window is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
PopupScreenWindow.ShowFocusVisual
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.StartupPosition

Description

The "StartupPosition" property specifies the position of the popup screen window at runtime start.

Type

Int32, HmiWindowStartupPosition

Specifies the position of the screen window:

- None (0): Relative placement on the configured monitor via "Left" and "Top".
- CenteredMonitor (1): Centered on the configured monitor.

- Maximized (2): Maximized on the configured monitor.
- CenteredOwner (3): Centered on the displayed screen.

Access

Read-write

Syntax

```
PopupScreenWindow.StartupPosition
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.StyleItemClass

Description

The "StyleItemClass" property returns the style applied to the popup screen window.

Type

String

Access

Read-only

Syntax

```
PopupScreenWindow.StyleItemClass
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.System

Description

The "System" property specifies the server prefix.

Type

String, HmiSystem

Access

Read-write

Syntax

```
PopupScreenWindow.System
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.TabIndex

Description

The "TabIndex" property returns the position of the popup screen window in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
PopupScreenWindow.TabIndex
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.TabIntoWindow

Description

The "TabIntoWindow" property specifies that the configured tab sequence of the displayed screen is resumed on activation via the configured tab sequence.

Type

Bool

Access

Read-write

Syntax

```
PopupScreenWindow.TabIntoWindow
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Top

Description

The "Top" property specifies the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
PopupScreenWindow.Top
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.VerticalScrollBarPosition

Description

The "VerticalScrollBarPosition" property specifies the vertical position for the scroll bar.

Type

Int32

Access

Read-write

Syntax

```
PopupScreenWindow.VerticalScrollBarPosition
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the window.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
PopupScreenWindow.VerticalScrollBarVisibility
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Visible

Description

The "Visible" property specifies whether the popup screen window is visible.

Type

Bool

Access

Read-write

Syntax

```
PopupScreenWindow.Visible
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Width

Description

The "Width" property specifies the width of the popup screen window in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
PopupScreenWindow.Width
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.WindowFlags

Description

The "WindowFlags" property specifies the screen window configuration.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
PopupScreenWindow.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the popup screen window.

Syntax

```
PopupScreenWindow.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.Close()

Description

The "Close" method closes the popup screen window.

Syntax

```
PopupScreenWindow.Close()
```

Parameters

--

Return value

Bool

See also

PopupScreenWindow (Page 4288)

PopupScreenWindow.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing means the change between two values of a property.

Syntax

```
PopupScreenWindow.PropertyFlashing(propertyName,enable[,value]
[,alternateValue][,rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flashing frequency:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

PopupScreenWindow (Page 4288)

ProcessControl

Description

The "ProcessControl" object shows a process control of tag values from the current process or the log in runtime.

Object type

HmiProcessControl

Properties

The "ProcessControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the process control.
- **EditMode**
  Specifies the editing mode for values in runtime.
- **Enabled**
  Specifies whether the process control can be operated in runtime.

- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the process control.
- **Layer**
  Returns the screen layer in which the process control is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the process control.
- **Online**
  Specifies the start and stop of updating the process control.
- **Parent**
  Returns the higher-level screen object.
- **ProcessView**
  Specifies the appearance of the process control.
- **RenderingTemplate**
  Returns the name of the template from which the process control was created.
- **ShowFocusVisual**
  Specifies whether the process control is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the process control.
- **StyleItemClass**
  Returns the style which is applied to the process control.
- **TabIndex**
  Returns the position of the process control in the tab sequence.
- **TimeStepSmoothingBase**
  Specifies the base of the time range for the smoothing the values/times.
- **TimeStepSmoothingFactor**
  Specifies the time base factor for smoothing.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Specifies the toolbar of the process control.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the process control is visible.

- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the process control.

Methods

The "ProcessControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the process control.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ProcessControl" object has the following events:

- **OnActivated()**
  Occurs when a process control receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the process control.
- **OnDeactivated()**
  Occurs when a process control loses focus.
- **OnInitialized()**
  Occurs when a process control has been successfully initialized and the data connection to the PLC has been established.

ProcessControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessControl.BackColor
```

See also

ProcessControl (Page 4322)

ProcessControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessControl.Caption
```

See also

ProcessControl (Page 4322)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessControl.Caption (Page 4325)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 4325)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 4325)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 4325)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 4325)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 4325)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 4325)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessControl.Caption (Page 4325)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessControl.Caption (Page 4325)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessControl.Caption (Page 4325)

ProcessControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ProcessControl.CaptionColor
```

See also

ProcessControl (Page 4322)

ProcessControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the process control.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ProcessControl.CurrentQuality
```

See also

ProcessControl (Page 4322)

ProcessControl.EditMode

Description

The "EditMode" property specifies the editing mode for values in runtime.

Type

Int32, HmiEditMode

Specifies the editing mode:

- None (0): No access
- Update (1): Update values
- Create (2): Create values
- Delete (4): Delete values

Access

Read-write

Syntax

```
ProcessControl.EditMode
```

See also

ProcessControl (Page 4322)

ProcessControl.Enabled

Description

The "Enabled" property specifies whether the process control can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessControl.Enabled
```

See also

ProcessControl (Page 4322)

ProcessControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessControl.Height
```

See also

ProcessControl (Page 4322)

ProcessControl.Icon

Description

The "Icon" property specifies the icon of the process control.

Type

String

Access

Read-write

Syntax

```
ProcessControl.Icon
```

See also

ProcessControl (Page 4322)

ProcessControl.Layer

Description

The "Layer" property returns the screen layer in which the process control is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ProcessControl.Layer
```

See also

ProcessControl (Page 4322)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ProcessControl.Layer (Page 4334)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ProcessControl.Layer (Page 4334)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ProcessControl.Layer (Page 4334)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ProcessControl.Layer (Page 4334)

ProcessControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessControl.Left
```

See also

ProcessControl (Page 4322)

ProcessControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ProcessControl.Margin
```

See also

ProcessControl (Page 4322)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ProcessControl.Margin (Page 4337)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ProcessControl.Margin (Page 4337)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ProcessControl.Margin (Page 4337)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ProcessControl.Margin (Page 4337)

ProcessControl.Name

Description

The "Name" property returns the name of the process control.

Type

String

Access

Read-only

Syntax

```
ProcessControl.Name
```

See also

ProcessControl (Page 4322)

ProcessControl.Online

Description

The "Online" property specifies the start and stop of the table view updating.

- True: Online. The table view is updated with new values.
- False: Offline. No new values are added to the table view.

Type

Bool

Access

Read-write

Syntax

```
ProcessControl.Online
```

See also

ProcessControl (Page 4322)

ProcessControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ProcessControl.Parent
```

See also

ProcessControl (Page 4322)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ProcessControl.ProcessView

Description

The "ProcessView" specifies the appearance of the table display.

Type

Object, HmiDataGridViewPart (Page 4341)

Access

Read-write

Syntax

```
ProcessControl.ProcessView
```

See also

ProcessControl (Page 4322)

DataGridView (Page 4341)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

ProcessControl.ProcessView (Page 4341)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 4344)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 4344)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 4344)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 4344)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 4348)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

ProcessControl.ProcessView (Page 4341)

HmiDataGridColumnCollection (Page 4348)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

See also

DataGridView.Columns (Page 4347)

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 4348)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 4350)

See also

HmiDataGridColumnCollection (Page 4348)

Process View Columns (Page 4350)

Process View Columns

ProcessColumn

Description

The "ProcessColumn" object represents a value column.

Object type

HmiProcessColumnPart

Properties

The "ProcessColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies the display options for text and graphics.
- **DataSource**
  Specifies the data source.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.

- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection (Page 4348)

ProcessColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
ProcessColumn.AllowSort
```

See also

ProcessColumn (Page 4350)

ProcessColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessColumn.BackColor
```

See also

ProcessColumn (Page 4350)

ProcessColumn.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ProcessColumn.Content
```

See also

ProcessColumn (Page 4350)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics

- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ProcessColumn.Content (Page 4352)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ProcessColumn.Content (Page 4352)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ProcessColumn.Content (Page 4352)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ProcessColumn.Content (Page 4352)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ProcessColumn.Content (Page 4352)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ProcessColumn.Content (Page 4352)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ProcessColumn.Content (Page 4352)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ProcessColumn.Content (Page 4352)

ProcessColumn.DataSource

Description

The "DataSource" property specifies the data source.

Type

Object, HmiDataSourcePart

Access

Read-write

Syntax

```
ProcessColumn.DataSource
```

See also

ProcessColumn (Page 4350)

DataSource.Source

Description

The "Source" property specifies the data source, e.g. a tag or logging tag.

Type

String

Access

Read-write

Syntax

```
DataSource.Source
```

See also

ProcessColumn.DataSource (Page 4357)

DataSource.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
DataSource.VisualizeQuality
```

See also

ProcessColumn.DataSource (Page 4357)

ProcessColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessColumn.Enabled
```

See also

ProcessColumn (Page 4350)

ProcessColumn.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessColumn.ForeColor
```

See also

ProcessColumn (Page 4350)

ProcessColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
ProcessColumn.Header
```

See also

ProcessColumn (Page 4350)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

ProcessColumn.Header (Page 4360)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 4360)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4360)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

ProcessColumn.Header (Page 4360)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

ProcessColumn.Header (Page 4360)

ProcessColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ProcessColumn.MaximumWidth
```

See also

ProcessColumn (Page 4350)

ProcessColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ProcessColumn.MinimumWidth
```

See also

ProcessColumn (Page 4350)

ProcessColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
ProcessColumn.Name
```

See also

ProcessColumn (Page 4350)

ProcessColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
ProcessColumn.OutputFormat
```

See also

ProcessColumn (Page 4350)

ProcessColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
ProcessColumn.SortDirection
```

See also

ProcessColumn (Page 4350)

ProcessColumn.SortOrder

Description

The "SortOrder" property specifies the sorting order.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
ProcessColumn.SortOrder
```

See also

ProcessColumn (Page 4350)

ProcessColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessColumn.Visible
```

See also

ProcessColumn (Page 4350)

ProcessColumn.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessColumn.Width
```

See also

ProcessColumn (Page 4350)

TimeRangeColumn

Description

The "TimeRangeColumn" object represents a value column.

Object type

HmiTimeRangeColumnPart

Properties

The "TimeRangeColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **BeginTime**
  Specifies the date and time for the start time of the time range.
- **Content**
  Specifies the display options for text and graphics.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **EndTime**
  Specifies the date and time for the end time of the time range.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **PointCount**
  Specifies the number of measurement points from the start time.
- **RangeType**
  Specifies the type of time range.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **TimeRangeBase**
  Specifies the basis of the time range.
- **TimeRangeFactor**
  Specifies the factor of the time base for defining the time range.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection (Page 4348)

TimeRangeColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
TimeRangeColumn.AllowSort
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TimeRangeColumn.BackColor
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.BeginTime

Description

The "BeginTime" property specifies the date and time for the start time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
TimeRangeColumn.BeginTime
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
TimeRangeColumn.Content
```

See also

TimeRangeColumn (Page 4370)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

TimeRangeColumn.Content (Page 4373)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

TimeRangeColumn.Content (Page 4373)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

TimeRangeColumn.Content (Page 4373)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

TimeRangeColumn.Content (Page 4373)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

TimeRangeColumn.Content (Page 4373)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

TimeRangeColumn.Content (Page 4373)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

TimeRangeColumn.Content (Page 4373)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

TimeRangeColumn.Content (Page 4373)

TimeRangeColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TimeRangeColumn.Enabled
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.EndTime

Description

The "EndTime" property specifies the date and time for the end time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
TimeRangeColumn.EndTime
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
TimeRangeColumn.ForeColor
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
TimeRangeColumn.Header
```

See also

TimeRangeColumn (Page 4370)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

TimeRangeColumn.Header (Page 4380)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 4380)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4380)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

TimeRangeColumn.Header (Page 4380)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

TimeRangeColumn.Header (Page 4380)

TimeRangeColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
TimeRangeColumn.MaximumWidth
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
TimeRangeColumn.MinimumWidth
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
TimeRangeColumn.Name
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
TimeRangeColumn.OutputFormat
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.PointCount

Description

The "PointCount" property specifies the number of measurement points from the start time.

Type

Int32

Access

Read-write

Syntax

```
TimeRangeColumn.PointCount
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.RangeType

Description

The "RangeType" property specifies the type of time range.

Type

Int32, HmiTimeRangeType

Specifies the time range:

- TimeRange (0): Any time range
- FromBeginToEnd (1): Total time range
- PointCount (2): Number of measurement points

Access

Read-write

Syntax

```
TimeRangeColumn.RangeType
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
TimeRangeColumn.SortDirection
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.SortOrder

Description

The "SortOrder" property specifies the sorting order.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
TimeRangeColumn.SortOrder
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.TimeRangeBase

Description

The "TimeRangeBase" property specifies the base of the time range.

Type

Int32, HmiTimeRangeBase

Specifies the time range:

- Undefined (0): Not defined
- Millisecond (1): Millisecond
- Second (2): Second
- Minute (3): Minute
- Hour (4): Hour
- Day (5): Day
- Month (6): Month
- Year (7): Year

Access

Read-write

Syntax

```
TimeRangeColumn.TimeRangeBase
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.TimeRangeFactor

Description

The "TimeRangeFactor" property specifies the factor of the time base for defining the time range.

Type

Int32

Access

Read-write

Syntax

```
TimeRangeColumn.TimeRangeFactor
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
TimeRangeColumn.Visible
```

See also

TimeRangeColumn (Page 4370)

TimeRangeColumn.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TimeRangeColumn.Width
```

See also

TimeRangeColumn (Page 4370)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

ProcessControl.ProcessView (Page 4341)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 4393)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 4393)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 4393)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 4393)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 4393)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 4393)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 4399)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 4401)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 4401)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 4401)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 4401)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 4401)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 4401)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 4399)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 4399)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

ProcessControl.ProcessView (Page 4341)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

ProcessControl.ProcessView (Page 4341)

ProcessControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the process control was created.

Type

String

Access

Read-only

Syntax

```
ProcessControl.RenderingTemplate
```

See also

ProcessControl (Page 4322)

ProcessControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the process control is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ProcessControl.ShowFocusVisual
```

See also

ProcessControl (Page 4322)

ProcessControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the process control.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
ProcessControl.StatusBar
```

See also

ProcessControl (Page 4322)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

ProcessControl.StatusBar (Page 4413)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 4415)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

ProcessControl.StatusBar (Page 4413)

HmiControlBarElementCollection (Page 4415)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 4414)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4415)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4433)

See also

HmiControlBarElementCollection (Page 4415)

Control Bar Elements (Page 4433)

Control Bar Elements

Description

Control Bar Elements (Page 4433)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

ProcessControl.StatusBar (Page 4413)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

ProcessControl.StatusBar (Page 4413)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 4417)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 4417)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 4417)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 4417)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 4417)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 4417)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

ProcessControl.StatusBar (Page 4413)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 4421)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 4421)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 4421)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 4421)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

ProcessControl.StatusBar (Page 4413)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 4424)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 4424)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 4424)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 4424)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

ProcessControl.StatusBar (Page 4413)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

ProcessControl.StatusBar (Page 4413)

ProcessControl.StyleItemClass

Description

The "StyleItemClass" property returns the style that is applied to the process control.

Type

String

Access

Read-only

Syntax

```
ProcessControl.StyleItemClass
```

See also

ProcessControl (Page 4322)

ProcessControl.TabIndex

Description

The "TabIndex" property returns the position of the process control in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ProcessControl.TabIndex
```

See also

ProcessControl (Page 4322)

ProcessControl.TimeStepSmoothingBase

Description

The "TimeStepSmoothingBase" property specifies the base of the time range for smoothing the values/times.

Type

Int32, HmiTimeRangeBase

Specifies the time range:

- Undefined (0): Not defined
- Millisecond (1): Millisecond
- Second (2): Second
- Minute (3): Minute
- Hour (4): Hour
- Day (5): Day
- Month (6): Month
- Year (7): Year

Access

Read-write

Syntax

```
ProcessControl.TimeStepSmoothingBase
```

See also

ProcessControl (Page 4322)

ProcessControl.TimeStepSmoothingFactor

Description

The "TimeStepSmoothingFactor" property specifies the factor for smoothing the time base.

Type

Int32

Access

Read-write

Syntax

```
ProcessControl.TimeStepSmoothingFactor
```

See also

ProcessControl (Page 4322)

ProcessControl.TimeZone

Description

The "TimeZone" property specifies the time zone.

Type

Int32, HmiTimeZone

Access

Read-write

Syntax

```
ProcessControl.TimeZone
```

See also

ProcessControl (Page 4322)

ProcessControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the table display.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
ProcessControl.ToolBar
```

See also

ProcessControl (Page 4322)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

ProcessControl.ToolBar (Page 4430)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 4431)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

HmiControlBarElementCollection (Page 4431)

ProcessControl.ToolBar (Page 4430)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 4431)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4431)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4433)

See also

HmiControlBarElementCollection (Page 4431)

Control Bar Elements (Page 4433)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 4433)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 4433)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 4433)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 4433)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 4433)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 4439)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 4439)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 4439)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 4439)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 4439)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 4439)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 4439)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 4439)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 4433)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 4433)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 4433)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 4449)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 4449)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 4449)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 4449)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 4433)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 4433)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 4433)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 4433)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 4454)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 4454)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 4454)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 4454)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 4433)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 4433)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 4433)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 4459)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 4461)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 4461)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4461)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 4461)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 4461)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 4461)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 4461)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4461)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 4459)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 4471)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 4471)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 4471)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 4471)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 4459)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 4476)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 4476)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 4476)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 4476)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 4459)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 4459)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.

- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms

- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export

- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 4481)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 4488)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 4488)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 4488)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 4488)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 4481)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 4493)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 4493)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 4493)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 4493)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 4481)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 4481)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 4499)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 4505)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 4505)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 4505)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 4505)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 4499)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 4510)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 4510)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 4510)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 4510)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 4499)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 4499)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.

- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 4515)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 4524)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 4524)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 4524)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 4524)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 4515)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 4529)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 4529)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 4529)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 4529)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 4515)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 4515)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.

- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 4535)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 4541)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 4541)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 4535)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 4552)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 4552)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 4552)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 4552)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 4535)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 4557)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 4557)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 4557)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 4557)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 4535)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 4535)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

ProcessControl.ToolBar (Page 4430)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

ProcessControl.ToolBar (Page 4430)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 4563)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 4563)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 4563)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 4563)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 4563)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 4563)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

ProcessControl.ToolBar (Page 4430)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 4567)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 4567)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 4567)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 4567)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

ProcessControl.ToolBar (Page 4430)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 4569)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 4569)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 4569)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 4569)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

ProcessControl.ToolBar (Page 4430)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

ProcessControl.ToolBar (Page 4430)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

ProcessControl.ToolBar (Page 4430)

ProcessControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessControl.Top
```

See also

ProcessControl (Page 4322)

ProcessControl.Visible

Description

The "Visible" property specifies whether the process control is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessControl.Visible
```

See also

ProcessControl (Page 4322)

ProcessControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessControl.Width
```

See also

ProcessControl (Page 4322)

ProcessControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the process control display.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
ProcessControl.WindowFlags
```

See also

ProcessControl (Page 4322)

ProcessControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the table view.

Syntax

```
ProcessControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ProcessControl (Page 4322)

ProcessControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the table view.

Syntax

```
ProcessControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

ProcessControl (Page 4322)

ProcessControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ProcessControl.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ProcessControl (Page 4322)

ProcessControl_OnActivated()

Description

The "OnActivated" event occurs when a table view receives focus:

- A table display is selected via the configured tab sequence.
- A table display that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
ProcessControl_OnActivated(item)
```

Context

item

Type: Object

Table view where the event occurs.

See also

ProcessControl (Page 4322)

ProcessControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the table view.

Syntax

```
ProcessControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Table view where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

ProcessControl (Page 4322)

ProcessControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when the table view loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
ProcessControl_OnDeactivated(item)
```

Context

item

Type: Object

Table view where the event occurs.

See also

ProcessControl (Page 4322)

ProcessControl_OnInitialized()

Description

The "OnInitialized" event occurs when a table view has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
ProcessControl_OnInitialized(item)
```

Context

item

Type: Object

Table view where the event occurs.

See also

ProcessControl (Page 4322)

ProcessDiagnosisCriteriaAnalysisControl

Description

The "ProcessDiagnosisCriteriaAnalysisControl" object represents a process diagnostics view for criteria analysis in runtime.

Object type

HmiProcessDiagnosisCriteriaAnalysisControl

Properties

The "ProcessDiagnosisCriteriaAnalysisControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CriteriaAnalysisView**
  Defines the appearance of the process diagnostics view.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the process diagnostics view.
- **Enabled**
  Specifies whether the process diagnostics view can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the process diagnostics view.
- **Layer**
  Returns the screen layer in which the process diagnostics view is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the process diagnostics view.

- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the process diagnostics view was created.
- **ShowFocusVisual**
  Specifies whether the process diagnostics view is highlighted when in focus.
- **SnapToSourceControl**
  Specifies whether the process diagnostics overview snaps to the window of the associated data source.
- **SourceControl**
  Specifies the data source.
- **StatusBar**
  Specifies the information bar of the process diagnostics view.
- **StyleItemClass**
  Returns the style which is applied to the process diagnostics view.
- **TabIndex**
  Returns the position of the process diagnostics view in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the process diagnostics view.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the process diagnostics view is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the process diagnostics view.

Methods

The "ProcessDiagnosisCriteriaAnalysisControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the process diagnostics view.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

ProcessDiagnosisCriteriaAnalysisControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.BackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Caption
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Caption (Page 4583)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 4583)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 4583)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 4583)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 4583)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 4583)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 4583)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Caption (Page 4583)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Caption (Page 4583)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Caption (Page 4583)

ProcessDiagnosisCriteriaAnalysisControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.CaptionColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView

Description

The "CriteriaAnalysisView" property specifies the appearance of the process diagnostics view.

Type

Object, HmiDataGridViewPart (Page 4590)

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

DataGridView (Page 4590)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 4592)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 4592)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 4592)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 4592)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 4596)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

HmiDataGridColumnCollection (Page 4596)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

See also

DataGridView.Columns (Page 4595)

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 4596)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 4598)

See also

HmiDataGridColumnCollection (Page 4596)

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn

Description

The "ProcessDiagnosisCriteriaAnalysisControlColumn" object represents a value column.

Object type

HmiProcessDiagnosisCriteriaAnalysisControlColumnPart

Properties

The "ProcessDiagnosisCriteriaAnalysisControlColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies display options for text and graphics.
- **CriteriaAnalysisBlock**
  Returns the criteria analysis block.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.

- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection (Page 4596)

ProcessDiagnosisCriteriaAnalysisControlColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.AllowSort
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.BackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.Content
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above

- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Content (Page 4600)

ProcessDiagnosisCriteriaAnalysisControlColumn.CriteriaAnalysisBlock

Description

The "CriteriaAnalysisBlock" property returns the criteria analysis block.

Type

Int32, HmiProcessDiagnosisCriteriaAnalysisBlock

Returns the criteria analysis block:

- SymbolName (0): Symbol name
- Address (1): Address
- Value (2): Value
- Comment (3): Comment

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.CriteriaAnalysisBlock
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.Enabled
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.ForeColor
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.Header
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Header (Page 4607)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 4607)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4607)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Header (Page 4607)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn.Header (Page 4607)

ProcessDiagnosisCriteriaAnalysisControlColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.MaximumWidth
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.MinimumWidth
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.Name
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.OutputFormat
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.SortDirection
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.SortOrder

Description

The "SortOrder" property specifies the order of the sorting.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.SortOrder
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.Visible
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

ProcessDiagnosisCriteriaAnalysisControlColumn.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControlColumn.Width
```

See also

ProcessDiagnosisCriteriaAnalysisControlColumn (Page 4598)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 4617)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 4617)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 4617)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 4617)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 4617)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 4617)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 4624)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 4626)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 4626)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 4626)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 4626)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 4626)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 4626)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 4624)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 4624)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.SelectionBorderWidth

Description

The "SelectionBorderWidth" property specifies the border width of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderWidth
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

ProcessDiagnosisCriteriaAnalysisControl.CriteriaAnalysisView (Page 4589)

ProcessDiagnosisCriteriaAnalysisControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the process diagnostics view.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.CurrentQuality
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Enabled

Description

The "Enabled" property specifies whether the process diagnostics view can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Enabled
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.ForeColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Height
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Icon

Description

The "Icon" property specifies the icon of the process diagnostics view.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Icon
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Layer

Description

The "Layer" property returns the screen layer in which the process diagnostics view is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Layer
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Layer (Page 4640)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Layer (Page 4640)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Layer (Page 4640)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Layer (Page 4640)

ProcessDiagnosisCriteriaAnalysisControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Left
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Margin
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Margin (Page 4643)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Margin (Page 4643)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Margin (Page 4643)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ProcessDiagnosisCriteriaAnalysisControl.Margin (Page 4643)

ProcessDiagnosisCriteriaAnalysisControl.Name

Description

The "Name" property returns the name of the process diagnostics view.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Name
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Parent
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ProcessDiagnosisCriteriaAnalysisControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the process diagnostics view was created.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.RenderingTemplate
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the process diagnostics view is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.ShowFocusVisual
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.SnapToSourceControl

Description

The "SnapToSourceControl" property specifies whether the process diagnostics overview snaps to the window of the associated data source.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.SnapToSourceControl
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.SourceControl

Description

The "SourceControl" property specifies the data source.

Type

Object, HmiAlarmControl (Page 1571)

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.SourceControl
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

AlarmControl (Page 1571)

AlarmControl

Description

AlarmControl (Page 1571)

ProcessDiagnosisCriteriaAnalysisControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the process diagnostics view.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.StatusBar
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 4650)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

HmiControlBarElementCollection (Page 4650)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 4649)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4650)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4667)

See also

HmiControlBarElementCollection (Page 4650)

Control Bar Elements (Page 4667)

Control Bar Elements

Description

Control Bar Elements (Page 4667)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 4653)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 4653)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 4653)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 4653)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 4653)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 4653)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 4656)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 4656)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 4656)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 4656)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 4659)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 4659)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 4659)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 4659)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

ProcessDiagnosisCriteriaAnalysisControl.StatusBar (Page 4648)

ProcessDiagnosisCriteriaAnalysisControl.StyleItemClass

Description

The "StyleItemClass" property returns the style that is applied to the process diagnostics view.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.StyleItemClass
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.TabIndex

Description

The "TabIndex" property returns the position of the process diagnostics view in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.TabIndex
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the process diagnostics view.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.ToolBar
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 4665)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

HmiControlBarElementCollection (Page 4665)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 4664)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4665)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4667)

See also

HmiControlBarElementCollection (Page 4665)

Control Bar Elements (Page 4667)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.

- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 4667)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 4667)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 4667)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 4667)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 4667)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 4672)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 4672)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 4672)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 4672)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 4672)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 4672)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 4672)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 4672)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 4667)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 4667)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 4667)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 4683)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 4683)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 4683)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 4683)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 4667)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 4667)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 4667)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 4667)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 4688)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 4688)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 4688)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 4688)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 4667)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 4667)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 4667)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.

- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 4693)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 4695)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 4695)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4695)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 4695)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 4695)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 4695)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 4695)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4695)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 4693)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 4705)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 4705)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 4705)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 4705)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 4693)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 4710)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 4710)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 4710)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 4710)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 4693)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 4693)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 4715)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 4722)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 4722)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 4722)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 4722)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 4715)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 4727)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 4727)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 4727)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 4727)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 4715)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 4715)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 4732)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 4738)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 4738)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 4738)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 4738)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 4732)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 4743)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 4743)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 4743)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 4743)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 4732)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 4732)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 4748)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 4757)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 4757)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 4757)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 4757)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 4748)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 4762)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 4762)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 4762)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 4762)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 4748)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 4748)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 4768)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 4775)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 4775)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 4768)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 4786)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 4786)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 4786)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 4786)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 4768)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 4791)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 4791)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 4791)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 4791)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 4768)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 4768)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 4797)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 4797)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 4797)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 4797)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 4797)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 4797)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 4800)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 4800)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 4800)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 4800)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 4803)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 4803)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 4803)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 4803)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

ProcessDiagnosisCriteriaAnalysisControl.ToolBar (Page 4663)

ProcessDiagnosisCriteriaAnalysisControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Top
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Visible

Description

The "Visible" property specifies whether the process diagnostics view is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Visible
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.Width
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the process diagnostics view.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title

- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.WindowFlags
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the process diagnostics view.

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the process diagnostics view for criteria analysis.

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.FireCommand(commandId,
custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisCriteriaAnalysisControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ProcessDiagnosisCriteriaAnalysisControl.PropertyFlashing(propertyNam
e, enable[, value][, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ProcessDiagnosisCriteriaAnalysisControl (Page 4581)

ProcessDiagnosisGraphOverviewControl

Description

The "ProcessDiagnosisGraphOverviewControl" object represents a process diagnostics view for graphic overview in runtime.

Object type

HmiProcessDiagnosisGraphOverviewControl

Properties

The "ProcessDiagnosisGraphOverviewControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the process diagnostics view.
- **Enabled**
  Specifies whether the process diagnostics view can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **GraphOverview**
  Defines the appearance of the process diagnostics view.
- **GridLineColor**
  Specifies the color of the grid lines.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the process diagnostics view.

- **Layer**
  Returns the screen layer in which the process diagnostics view is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the process diagnostics view.
- **OperationMode**
  Specifies the operating mode.
- **Parent**
  Returns the higher-level screen object.
- **PlcSource**
  Specifies the PLC source.
- **RenderingTemplate**
  Returns the name of the template from which the process diagnostics view was created.
- **SelectedStep**
  Specifies which step is selected.
- **ShowFocusVisual**
  Specifies whether the process diagnostics view is highlighted when in focus.
- **ShowOperationMode**
  Specifies whether to display the operating mode.
- **StatusBar**
  Specifies the information bar of the process diagnostics view.
- **StyleItemClass**
  Returns the style which is applied to the process diagnostics view.
- **TabIndex**
  Returns the position of the process diagnostics view in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the process diagnostics view.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the process diagnostics view is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the process diagnostics view.

Methods

The "ProcessDiagnosisGraphOverviewControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the process diagnostics view.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ProcessDiagnosisGraphOverviewControl" object has the following events:

- **AlarmViewButtonTapped()**
  Occurs when the "AlarmView" button is pressed.
- **PlcCodeViewerButtonTapped()**
  Occurs when the "PlcCodeViewer" button is pressed.
- **TIAPortalButtonTapped()**
  Occurs when the "TIAPortal" button is pressed.

ProcessDiagnosisGraphOverviewControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.BackColor
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Caption
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessDiagnosisGraphOverviewControl.Caption (Page 4815)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 4815)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 4815)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 4815)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 4815)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 4815)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 4815)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessDiagnosisGraphOverviewControl.Caption (Page 4815)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessDiagnosisGraphOverviewControl.Caption (Page 4815)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessDiagnosisGraphOverviewControl.Caption (Page 4815)

ProcessDiagnosisGraphOverviewControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.CaptionColor
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the process diagnostics view.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.CurrentQuality
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Enabled

Description

The "Enabled" property specifies whether the process diagnostics view can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Enabled
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Font
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ProcessDiagnosisGraphOverviewControl.Font (Page 4822)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ProcessDiagnosisGraphOverviewControl.Font (Page 4822)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ProcessDiagnosisGraphOverviewControl.Font (Page 4822)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ProcessDiagnosisGraphOverviewControl.Font (Page 4822)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ProcessDiagnosisGraphOverviewControl.Font (Page 4822)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ProcessDiagnosisGraphOverviewControl.Font (Page 4822)

ProcessDiagnosisGraphOverviewControl.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.ForeColor
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.GraphOverview

Description

The "GraphOverview" property specifies the appearance of the process diagnostics view.

Type

Object, HmiDataGridViewPart (Page 4827)

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.GraphOverview
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

DataGridView (Page 4827)

DataGridView

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 4829)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 4829)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 4829)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 4829)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 4833)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

HmiDataGridColumnCollection (Page 4833)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

See also

DataGridView.Columns (Page 4833)

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 4833)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 4835)

See also

HmiDataGridColumnCollection (Page 4833)

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn

Description

The "GraphOverviewControlColumn" object represents a value column.

Object type

HmiGraphOverviewControlColumnPart

Properties

The "GraphOverviewControlColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies display options for text and graphics.

- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **GraphOverviewControlBlock**
  Specifies the graphic overview block.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection.Item() (Page 4834)

GraphOverviewControlColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
GraphOverviewControlColumn.AllowSort
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
GraphOverviewControlColumn.BackColor
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
GraphOverviewControlColumn.Content
```

See also

GraphOverviewControlColumn (Page 4835)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

GraphOverviewControlColumn.Content (Page 4837)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

GraphOverviewControlColumn.Content (Page 4837)

GraphOverviewControlColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
GraphOverviewControlColumn.Enabled
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
GraphOverviewControlColumn.ForeColor
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.GraphOverviewControlBlock

Description

The "GraphOverviewControlBlock" property specifies the graphic overview block.

Type

Int32, HmiGraphOverviewBlock

Specifies the graphic overview block:

- Undefined (0): Not defined
- Step (1): Step
- StepName (2): Step name

Access

Read-write

Syntax

```
GraphOverviewControlColumn.GraphOverviewControlBlock
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
GraphOverviewControlColumn.Header
```

See also

GraphOverviewControlColumn (Page 4835)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

GraphOverviewControlColumn.Header (Page 4844)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 4845)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 4845)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

GraphOverviewControlColumn.Header (Page 4844)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

GraphOverviewControlColumn.Header (Page 4844)

GraphOverviewControlColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
GraphOverviewControlColumn.MaximumWidth
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
GraphOverviewControlColumn.MinimumWidth
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
GraphOverviewControlColumn.Name
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
GraphOverviewControlColumn.OutputFormat
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
GraphOverviewControlColumn.SortDirection
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.SortOrder

Description

The "SortOrder" property specifies the order of the sorting.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
GraphOverviewControlColumn.SortOrder
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
GraphOverviewControlColumn.Visible
```

See also

GraphOverviewControlColumn (Page 4835)

GraphOverviewControlColumn.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
GraphOverviewControlColumn.Width
```

See also

GraphOverviewControlColumn (Page 4835)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 4855)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 4855)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 4855)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 4855)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 4855)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 4855)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 4862)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 4864)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 4864)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 4864)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 4864)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 4864)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 4864)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 4862)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 4862)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.SelectionBorderWidth

Description

The "SelectionBorderWidth" property specifies the border width of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderWidth
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

ProcessDiagnosisGraphOverviewControl.GraphOverview (Page 4826)

ProcessDiagnosisGraphOverviewControl.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.GridLineColor
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Height
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Icon

Description

The "Icon" property specifies the icon of the process diagnostics view.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Icon
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Layer

Description

The "Layer" property returns the screen layer in which the process diagnostics view is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.Layer
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ProcessDiagnosisGraphOverviewControl.Layer (Page 4877)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ProcessDiagnosisGraphOverviewControl.Layer (Page 4877)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ProcessDiagnosisGraphOverviewControl.Layer (Page 4877)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ProcessDiagnosisGraphOverviewControl.Layer (Page 4877)

ProcessDiagnosisGraphOverviewControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Left
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Margin
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ProcessDiagnosisGraphOverviewControl.Margin (Page 4880)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ProcessDiagnosisGraphOverviewControl.Margin (Page 4880)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ProcessDiagnosisGraphOverviewControl.Margin (Page 4880)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ProcessDiagnosisGraphOverviewControl.Margin (Page 4880)

ProcessDiagnosisGraphOverviewControl.Name

Description

The "Name" property returns the name of the process diagnostics view.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.Name
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.OperationMode

Description

The "OperationMode" property specifies the operating mode.

Type

Object, HmiProcessDiagnosisOperationModePart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.OperationMode
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisOperationMode.OpModeAutoText

Description

The "OpModeAutoText" property specifies the automatic text.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisOperationMode.OpModeAutoText
```

See also

ProcessDiagnosisGraphOverviewControl.OperationMode (Page 4883)

ProcessDiagnosisOperationMode.OpModeManText

Description

The "OpModeManText" property specifies the manual text.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisOperationMode.OpModeManText
```

See also

ProcessDiagnosisGraphOverviewControl.OperationMode (Page 4883)

ProcessDiagnosisOperationMode.OpModeTapText

Description

The "OpModeTapText" property specifies the text on clicking.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisOperationMode.OpModeTapText
```

See also

ProcessDiagnosisGraphOverviewControl.OperationMode (Page 4883)

ProcessDiagnosisOperationMode.OpModeTopText

Description

The "OpModeTopText" property specifies the title text.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisOperationMode.OpModeTopText
```

See also

ProcessDiagnosisGraphOverviewControl.OperationMode (Page 4883)

ProcessDiagnosisGraphOverviewControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.Parent
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ProcessDiagnosisGraphOverviewControl.PlcSource

Description

The "PlcSource" property specifies the PLC source.

Type

Object, HmiPlcDataSourcePart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.PlcSource
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

PlcDataSource.Connection

Description

The "Connection" property returns the connection.

Type

Object, HmiConnection (Page 1148)

Access

Read-only

Syntax

```
PlcDataSource.Connection
```

See also

ProcessDiagnosisGraphOverviewControl.PlcSource (Page 4886)

Connection (Page 1148)

Connection

Description

Connection (Page 1148)

PlcDataSource.DB_Name

Description

The "DB_Name" property returns the name of the database.

Type

String

Access

Read-only

Syntax

```
PlcDataSource.DB_Name
```

See also

ProcessDiagnosisGraphOverviewControl.PlcSource (Page 4886)

PlcDataSource.HmiConnectionName

Description

The "HmiConnectionName" property specifies the connection name.

Type

String

Access

Read-write

Syntax

```
PlcDataSource.HmiConnectionName
```

See also

ProcessDiagnosisGraphOverviewControl.PlcSource (Page 4886)

PlcDataSource.Tag

Description

The "Tag" property returns the tag.

Type

String, HMITag (Page 1338)

Access

Read-only

Syntax

```
PlcDataSource.Tag
```

See also

ProcessDiagnosisGraphOverviewControl.PlcSource (Page 4886)

Tag (Page 1338)

Tag

Description

Tag (Page 1338)

ProcessDiagnosisGraphOverviewControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the process diagnostics view was created.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.RenderingTemplate
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.SelectedStep

Description

The "SelectedStep" property specifies which step is selected.

Type

UInt16

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.SelectedStep
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the process diagnostics view is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.ShowFocusVisual
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.ShowOperationMode

Description

The "ShowOperationMode" property specifies whether to display the operating mode.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.ShowOperationMode
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the process diagnostics view.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.StatusBar
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 4892)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

HmiControlBarElementCollection (Page 4892)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 4892)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4892)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4909)

See also

HmiControlBarElementCollection (Page 4892)

Control Bar Elements (Page 4909)

Control Bar Elements

Description

Control Bar Elements (Page 4909)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 4895)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 4895)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 4895)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 4895)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 4895)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 4895)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 4899)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 4899)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 4899)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 4899)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 4901)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 4901)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 4901)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 4901)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

ProcessDiagnosisGraphOverviewControl.StatusBar (Page 4891)

ProcessDiagnosisGraphOverviewControl.StyleItemClass

Description

The "StyleItemClass" property returns the style that is applied to the process diagnostics view.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.StyleItemClass
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.TabIndex

Description

The "TabIndex" property returns the position of the process diagnostics view in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ProcessDiagnosisGraphOverviewControl.TabIndex
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the process diagnostics view.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.ToolBar
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 4907)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

HmiControlBarElementCollection (Page 4907)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 4907)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 4907)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 4909)

See also

HmiControlBarElementCollection (Page 4907)

Control Bar Elements (Page 4909)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 4909)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 4909)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 4909)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 4909)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 4909)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 4915)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 4915)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 4915)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 4915)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 4915)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 4915)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 4915)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 4915)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 4909)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 4909)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 4909)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 4925)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 4925)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 4925)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 4925)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 4909)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 4909)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 4909)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 4909)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 4930)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 4930)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 4930)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 4930)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 4909)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 4909)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 4909)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 4935)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 4937)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 4937)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4937)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 4937)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 4937)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 4937)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 4937)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 4937)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 4935)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 4947)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 4947)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 4947)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 4947)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 4935)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 4952)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 4952)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 4952)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 4952)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 4935)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 4935)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.

- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms

- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export

- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 4957)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 4964)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 4964)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 4964)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 4964)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 4957)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 4969)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 4969)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 4969)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 4969)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 4957)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 4957)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 4975)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 4981)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 4981)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 4981)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 4981)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 4975)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 4986)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 4986)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 4986)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 4986)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 4975)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 4975)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.

- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 4991)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 5000)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 5000)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 5000)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 5000)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 4991)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 5005)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 5005)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 5005)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 5005)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 4991)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 4991)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.

- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 5011)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 5017)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5017)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 5011)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 5028)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 5028)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 5028)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 5028)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 5011)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 5033)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 5033)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 5033)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 5033)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 5011)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 5011)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 5039)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 5039)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 5039)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 5039)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 5039)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 5039)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 5043)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 5043)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 5043)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 5043)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 5045)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 5045)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 5045)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 5045)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

ProcessDiagnosisGraphOverviewControl.ToolBar (Page 4906)

ProcessDiagnosisGraphOverviewControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Top
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Visible

Description

The "Visible" property specifies whether the process diagnostics view is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Visible
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.Width
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the process diagnostics view.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
ProcessDiagnosisGraphOverviewControl.WindowFlags
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the process diagnostics view.

Syntax

```
ProcessDiagnosisGraphOverviewControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the process diagnostics view for graphic overview.

Syntax

```
ProcessDiagnosisGraphOverviewControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ProcessDiagnosisGraphOverviewControl.PropertyFlashing(propertyName,
enable[, value][, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl_AlarmViewButtonTapped()

Description

The "AlarmViewButtonTapped"event occurs when the "AlarmView" button is pressed.

Syntax

```
ProcessDiagnosisGraphOverviewControl_AlarmViewButtonTapped(item)
```

Context

`item`

Type: Object

Button at which the event occurs.

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl_PlcCodeViewerButtonTapped()

Description

The "PlcCodeViewerButtonTapped"event occurs when the "PlcCodeViewer" button is pressed.

Syntax

```
ProcessDiagnosisGraphOverviewControl_PlcCodeViewerButtonTapped(item)
```

Context

item

Type: Object

Button at which the event occurs.

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisGraphOverviewControl_TIAPortalButtonTapped()

Description

The "TIAPortalButtonTapped"event occurs when the "TIAPortal" button is pressed.

Syntax

```
ProcessDiagnosisGraphOverviewControl_TIAPortalButtonTapped(item)
```

Context

item

Type: Object

Button at which the event occurs.

See also

ProcessDiagnosisGraphOverviewControl (Page 4812)

ProcessDiagnosisOverviewControl

Description

The "ProcessDiagnosisOverviewControl" object represents a process diagnostics overview in runtime.

Object type

HmiProcessDiagnosisOverviewControl

Properties

The "Object" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the process diagnostics overview.
- **Enabled**
  Specifies whether the process diagnostics overview can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the process diagnostics overview.
- **Layer**
  Returns the screen layer in which the process diagnostics overview is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the process diagnostics overview.
- **Parent**
  Returns the higher-level screen object.

- **PDiagCategories**
  Specifies the category of the process diagnostics overview.
- **PDiagSupervisionTypes**
  Specifies the supervision types of the process diagnostics overview.
- **PlcSource**
  Specifies the PLC source.
- **RenderingTemplate**
  Returns the name of the template from which the process diagnostics overview was created.
- **ShowFocusVisual**
  Specifies whether the process diagnostics overview is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the process diagnostics overview.
- **StyleItemClass**
  Returns the style which is applied to the process diagnostics overview.
- **TabIndex**
  Returns the position of the process diagnostics overview in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the process diagnostics overview.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the process diagnostics overview is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the process diagnostics overview.

Methods

The "ProcessDiagnosisOverviewControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the process diagnostics overview.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ProcessDiagnosisOverviewControl" object has the following events:

- **AlarmViewButtonTapped()**
  Occurs when the "AlarmView" button is pressed.

ProcessDiagnosisOverviewControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.BackColor
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Caption
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessDiagnosisOverviewControl.Caption (Page 5058)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5059)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5059)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5059)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5059)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5059)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5059)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessDiagnosisOverviewControl.Caption (Page 5058)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessDiagnosisOverviewControl.Caption (Page 5058)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessDiagnosisOverviewControl.Caption (Page 5058)

ProcessDiagnosisOverviewControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.CaptionColor
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the process diagnostics overview.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.CurrentQuality
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Enabled

Description

The "Enabled" property specifies whether the process diagnostics overview can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Enabled
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Font
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ProcessDiagnosisOverviewControl.Font (Page 5066)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ProcessDiagnosisOverviewControl.Font (Page 5066)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ProcessDiagnosisOverviewControl.Font (Page 5066)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ProcessDiagnosisOverviewControl.Font (Page 5066)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ProcessDiagnosisOverviewControl.Font (Page 5066)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ProcessDiagnosisOverviewControl.Font (Page 5066)

ProcessDiagnosisOverviewControl.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.ForeColor
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Height
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Icon

Description

The "Icon" property specifies the icon of the process diagnostics overview.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Icon
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Layer

Description

The "Layer" property returns the screen layer in which the process diagnostics overview is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.Layer
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ProcessDiagnosisOverviewControl.Layer (Page 5071)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ProcessDiagnosisOverviewControl.Layer (Page 5071)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ProcessDiagnosisOverviewControl.Layer (Page 5071)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ProcessDiagnosisOverviewControl.Layer (Page 5071)

ProcessDiagnosisOverviewControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Left
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Margin
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ProcessDiagnosisOverviewControl.Margin (Page 5074)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ProcessDiagnosisOverviewControl.Margin (Page 5074)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ProcessDiagnosisOverviewControl.Margin (Page 5074)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ProcessDiagnosisOverviewControl.Margin (Page 5074)

ProcessDiagnosisOverviewControl.Name

Description

The "Name" property returns the name of the process diagnostics overview.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.Name
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.Parent
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ProcessDiagnosisOverviewControl.PDiagCategories

Description

The "PDiagCategories" property specifies the category of the process diagnostics overview.

Type

Object, HmiProcessDiagnosisOverviewPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.PDiagCategories
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverview.Label

Description

The "Label" property specifies the label.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverview.Label
```

See also

ProcessDiagnosisOverviewControl.PDiagCategories (Page 5077)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessDiagnosisOverview.Label (Page 5078)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5078)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5078)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5078)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5078)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5078)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5078)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessDiagnosisOverview.Label (Page 5078)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessDiagnosisOverview.Label (Page 5078)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessDiagnosisOverview.Label (Page 5078)

ProcessDiagnosisOverview.PDiagElements

Description

The "PDiagElements" property represents the quantity of elements.

Type

Object, HmiProcessDiagnosisOverviewElementCollection (Page 5084)

Access

Read-only

Syntax

```
ProcessDiagnosisOverview.PDiagElements
```

See also

ProcessDiagnosisOverviewControl.PDiagCategories (Page 5077)

HmiProcessDiagnosisOverviewElementCollection (Page 5084)

HmiProcessDiagnosisOverviewElementCollection

Description

The "HmiProcessDiagnosisOverviewElementCollection" object is a list of all process diagnostics overview elements ("ProcessDiagnosisOverviewElement" objects).

Use

The "HmiProcessDiagnosisOverviewElementCollection" object is a list and can be counted and enumerated. You can access the "HmiProcessDiagnosisOverviewElementCollection" list using the index or the tag names.

Object type

HmiProcessDiagnosisOverviewElementCollection

Properties

The "HmiProcessDiagnosisOverviewElementCollection" object has the following properties:

- **Count**
  Returns the number of process diagnostics overview elements in the "HmiProcessDiagnosisOverviewElementCollection" list.

Methods

The "HmiProcessDiagnosisOverviewElementCollection" object has the following methods:

- **Item()**
  Returns a process diagnostics overview element of the "HmiProcessDiagnosisOverviewElementCollection" list.

See also

ProcessDiagnosisOverview.PDiagElements (Page 5084)

HmiProcessDiagnosisOverviewElementCollection.Count

Description

The "Count" property returns the number of process diagnostics overview elements in the "HmiProcessDiagnosisOverviewElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiProcessDiagnosisOverviewElementCollection.Count
```

See also

HmiProcessDiagnosisOverviewElementCollection (Page 5084)

HmiProcessDiagnosisOverviewElementCollection.Item()

Description

The "Item" method returns a process diagnostics overview element of the "HmiProcessDiagnosisOverviewElementCollection" list.

Syntax

```
HmiProcessDiagnosisOverviewElementCollection[.Item]
(HmiProcessDiagnosisOverviewElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiProcessDiagnosisOverviewElementCollection" object.

Parameters

`HmiProcessDiagnosisOverviewElementName`

Type: String

Name of the process diagnostics overview element

Return value

Object, HmiProcessDiagnosisOverviewElementPart (Page 5086)

See also

HmiProcessDiagnosisOverviewElementCollection (Page 5084)

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement

Description

The "ProcessDiagnosisOverviewElement" object represents a process diagnostics overview element.

Object type

HmiProcessDiagnosisOverviewElementPart

Properties

The "ProcessDiagnosisOverviewElement" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateForeColor**
  Specifies the flashing color for the text.
- **BackColor**
  Specifies the background color.
- **FlashingRate**
  Specifies the flash rate.

- **ForeColor**
  Specifies the font color of the text.
- **Text**
  Specifies the labeling of the process diagnostics overview element.
- **Visible**
  Specifies whether the process diagnostics overview element is visible.

Methods

--

See also

HmiProcessDiagnosisOverviewElementCollection.Item() (Page 5085)

ProcessDiagnosisOverviewElement.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.AlternateBackColor
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.AlternateForeColor
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.BackColor
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement.FlashingRate

Description

The "FlashingRate" property specifies the flash rate.

Type

Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.FlashingRate
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.ForeColor
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement.Text

Description

The "Text" property specifies the labeling of the process diagnostics overview element.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.Text
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverviewElement.Visible

Description

The "Visible" property specifies whether the process diagnostics overview element is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.Visible
```

See also

ProcessDiagnosisOverviewElement (Page 5086)

ProcessDiagnosisOverview.SymbolFont

Description

The "SymbolFont" property specifies the font of the symbol text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverview.SymbolFont
```

See also

ProcessDiagnosisOverviewControl.PDiagCategories (Page 5077)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5090)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5090)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5090)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5090)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5090)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5090)

ProcessDiagnosisOverview.Visible

Description

The "Visible" property specifies whether the process diagnostics view is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverview.Visible
```

See also

ProcessDiagnosisOverviewControl.PDiagCategories (Page 5077)

ProcessDiagnosisOverviewControl.PDiagSupervisionTypes

Description

The "PDiagSupervisionTypes" property specifies the supervision types of the process diagnostics overview.

Type

Object, HmiProcessDiagnosisOverviewPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.PDiagSupervisionTypes
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverview.Label

Description

The "Label" property specifies the label.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverview.Label
```

See also

ProcessDiagnosisOverviewControl.PDiagSupervisionTypes (Page 5095)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessDiagnosisOverview.Label (Page 5095)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5096)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5096)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5096)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5096)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5096)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5096)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessDiagnosisOverview.Label (Page 5095)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessDiagnosisOverview.Label (Page 5095)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessDiagnosisOverview.Label (Page 5095)

ProcessDiagnosisOverview.PDiagElements

Description

The "PDiagElements" property represents the quantity of elements.

Type

Object, HmiProcessDiagnosisOverviewElementCollection (Page 5101)

Access

Read-only

Syntax

```
ProcessDiagnosisOverview.PDiagElements
```

See also

ProcessDiagnosisOverviewControl.PDiagSupervisionTypes (Page 5095)

HmiProcessDiagnosisOverviewElementCollection (Page 5101)

HmiProcessDiagnosisOverviewElementCollection

Description

The "HmiProcessDiagnosisOverviewElementCollection" object is a list of all process diagnostics overview elements ("ProcessDiagnosisOverviewElement" objects).

Use

The "HmiProcessDiagnosisOverviewElementCollection" object is a list and can be counted and enumerated. You can access the "HmiProcessDiagnosisOverviewElementCollection" list using the index or the tag names.

Object type

HmiProcessDiagnosisOverviewElementCollection

Properties

The "HmiProcessDiagnosisOverviewElementCollection" object has the following properties:

- **Count**
  Returns the number of process diagnostics overview elements in the "HmiProcessDiagnosisOverviewElementCollection" list.

Methods

The "HmiProcessDiagnosisOverviewElementCollection" object has the following methods:

- **Item()**
  Returns a process diagnostics overview element of the "HmiProcessDiagnosisOverviewElementCollection" list.

See also

ProcessDiagnosisOverview.PDiagElements (Page 5101)

HmiProcessDiagnosisOverviewElementCollection.Count

Description

The "Count" property returns the number of process diagnostics overview elements in the "HmiProcessDiagnosisOverviewElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiProcessDiagnosisOverviewElementCollection.Count
```

See also

HmiProcessDiagnosisOverviewElementCollection (Page 5101)

HmiProcessDiagnosisOverviewElementCollection.Item()

Description

The "Item" method returns a process diagnostics overview element of the "HmiProcessDiagnosisOverviewElementCollection" list.

Syntax

```
HmiProcessDiagnosisOverviewElementCollection[.Item]
(HmiProcessDiagnosisOverviewElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiProcessDiagnosisOverviewElementCollection" object.

Parameters

`HmiProcessDiagnosisOverviewElementName`

Type: String

Name of the process diagnostics overview element

Return value

Object, HmiProcessDiagnosisOverviewElementPart (Page 5103)

See also

HmiProcessDiagnosisOverviewElementCollection (Page 5101)

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement

Description

The "ProcessDiagnosisOverviewElement" object represents a process diagnostics overview element.

Object type

HmiProcessDiagnosisOverviewElementPart

Properties

The "ProcessDiagnosisOverviewElement" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateForeColor**
  Specifies the flashing color for the text.
- **BackColor**
  Specifies the background color.
- **FlashingRate**
  Specifies the flash rate.
- **ForeColor**
  Specifies the font color of the text.
- **Text**
  Specifies the labeling of the process diagnostics overview element.
- **Visible**
  Specifies whether the process diagnostics overview element is visible.

Methods

--

See also

HmiProcessDiagnosisOverviewElementCollection.Item() (Page 5103)

ProcessDiagnosisOverviewElement.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.AlternateBackColor
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.AlternateForeColor
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.BackColor
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement.FlashingRate

Description

The "FlashingRate" property specifies the flash rate.

Type

Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.FlashingRate
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.ForeColor
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement.Text

Description

The "Text" property specifies the labeling of the process diagnostics overview element.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.Text
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverviewElement.Visible

Description

The "Visible" property specifies whether the process diagnostics overview element is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewElement.Visible
```

See also

ProcessDiagnosisOverviewElement (Page 5103)

ProcessDiagnosisOverview.SymbolFont

Description

The "SymbolFont" property specifies the font of the symbol text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverview.SymbolFont
```

See also

ProcessDiagnosisOverviewControl.PDiagSupervisionTypes (Page 5095)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5108)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5108)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5108)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5108)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5108)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ProcessDiagnosisOverview.SymbolFont (Page 5108)

ProcessDiagnosisOverview.Visible

Description

The "Visible" property specifies whether the process diagnostics view is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverview.Visible
```

See also

ProcessDiagnosisOverviewControl.PDiagSupervisionTypes (Page 5095)

ProcessDiagnosisOverviewControl.PlcSource

Description

The "PlcSource" property specifies the PLC source.

Type

Object, HmiPlcDataSourcePart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.PlcSource
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

PlcDataSource.Connection

Description

The "Connection" property returns the connection.

Type

Object, HmiConnection (Page 1148)

Access

Read-only

Syntax

```
PlcDataSource.Connection
```

See also

ProcessDiagnosisOverviewControl.PlcSource (Page 5112)

Connection (Page 1148)

Connection

Description

Connection (Page 1148)

PlcDataSource.DB_Name

Description

The "DB_Name" property returns the name of the database.

Type

String

Access

Read-only

Syntax

```
PlcDataSource.DB_Name
```

See also

ProcessDiagnosisOverviewControl.PlcSource (Page 5112)

PlcDataSource.HmiConnectionName

Description

The "HmiConnectionName" property specifies the connection name.

Type

String

Access

Read-write

Syntax

```
PlcDataSource.HmiConnectionName
```

See also

ProcessDiagnosisOverviewControl.PlcSource (Page 5112)

PlcDataSource.Tag

Description

The "Tag" property returns the tag.

Type

String, HMITag (Page 1338)

Access

Read-only

Syntax

```
PlcDataSource.Tag
```

See also

ProcessDiagnosisOverviewControl.PlcSource (Page 5112)

Tag (Page 1338)

Tag

Description

Tag (Page 1338)

ProcessDiagnosisOverviewControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the process diagnostics overview was created.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.RenderingTemplate
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the process diagnostics overview is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.ShowFocusVisual
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the process diagnostics overview.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.StatusBar
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 5117)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

HmiControlBarElementCollection (Page 5117)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 5117)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 5117)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 5134)

See also

HmiControlBarElementCollection (Page 5117)

Control Bar Elements (Page 5134)

Control Bar Elements

Description

Control Bar Elements (Page 5134)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 5120)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 5120)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 5120)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 5120)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 5120)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 5120)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 5124)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 5124)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 5124)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 5124)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 5126)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 5126)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 5126)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 5126)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

ProcessDiagnosisOverviewControl.StatusBar (Page 5116)

ProcessDiagnosisOverviewControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the process diagnostics overview.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.StyleItemClass
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.TabIndex

Description

The "TabIndex" property returns the position of the process diagnostics overview in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ProcessDiagnosisOverviewControl.TabIndex
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.ToolBar

Description

The property "ToolBar" specifies the toolbar of the process diagnostics overview.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.ToolBar
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 5132)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

HmiControlBarElementCollection (Page 5132)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 5132)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 5132)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 5134)

See also

HmiControlBarElementCollection (Page 5132)

Control Bar Elements (Page 5134)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 5134)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 5134)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 5134)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 5134)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 5134)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 5140)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 5140)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 5140)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 5140)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 5140)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 5140)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 5140)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 5140)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 5134)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 5134)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 5134)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 5150)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 5150)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 5150)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 5150)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 5134)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 5134)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 5134)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 5134)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 5155)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 5155)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 5155)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 5155)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 5134)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 5134)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 5134)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 5160)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 5162)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 5162)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 5162)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 5162)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 5162)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 5162)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 5162)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 5162)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 5160)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 5172)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 5172)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 5172)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 5172)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 5160)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 5177)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 5177)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 5177)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 5177)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 5160)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 5160)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.

- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms

- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export

- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 5182)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 5189)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 5189)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 5189)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 5189)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 5182)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 5194)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 5194)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 5194)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 5194)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 5182)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 5182)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 5200)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 5206)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 5206)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 5206)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 5206)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 5200)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 5211)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 5211)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 5211)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 5211)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 5200)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 5200)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.

- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 5216)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 5225)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 5225)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 5225)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 5225)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 5216)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 5230)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 5230)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 5230)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 5230)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 5216)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 5216)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.

- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 5236)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 5242)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5242)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 5236)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 5253)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 5253)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 5253)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 5253)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 5236)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 5258)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 5258)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 5258)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 5258)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 5236)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 5236)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 5264)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 5264)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 5264)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 5264)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 5264)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 5264)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 5268)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 5268)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 5268)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 5268)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 5270)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 5270)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 5270)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 5270)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

ProcessDiagnosisOverviewControl.ToolBar (Page 5131)

ProcessDiagnosisOverviewControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Top
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Visible

Description

The "Visible" property specifies whether the process diagnostics view is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Visible
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.Width
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the process diagnostics overview.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
ProcessDiagnosisOverviewControl.WindowFlags
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the process diagnostics overview.

Syntax

```
ProcessDiagnosisOverviewControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the process diagnostics overview.

Syntax

```
ProcessDiagnosisOverviewControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ProcessDiagnosisOverviewControl.PropertyFlashing(propertyName,
enable[, value][, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisOverviewControl_AlarmViewButtonTapped()

Description

The "AlarmViewButtonTapped"event occurs when the "AlarmView" button is pressed.

Syntax

```
ProcessDiagnosisOverviewControl_AlarmViewButtonTapped(item)
```

Context

item

Type: Object

Button at which the event occurs.

See also

ProcessDiagnosisOverviewControl (Page 5056)

ProcessDiagnosisPlcCodeViewerControl

Description

The "ProcessDiagnosisPlcCodeViewerControl" object represents a PLC code view in runtime.

Object type

HmiProcessDiagnosisPlcCodeViewerControl

Properties

The "ProcessDiagnosisPlcCodeViewerControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the PLC code view.
- **CurrentZoomFactor**
  Specifies the zoom factor of the PLC code view.
- **Enabled**
  Specifies whether the PLC code view can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the PLC code view.
- **Layer**
  Returns the screen layer in which the PLC code view is located.
- **Left**
  Specifies the value of the X coordinate.
- **LineColors**
  Returns the line colors.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the PLC code view.

- **OverviewDetailRatio**
  Specifies how much space the detail view takes up in the PLC code view.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the PLC code view was created.
- **ShowFocusVisual**
  Specifies whether the PLC code view is highlighted when in focus.
- **ShowSymbolLine**
  Specifies whether the symbol line is displayed.
- **StatusBar**
  Specifies the information bar of the PLC code view.
- **StyleItemClass**
  Returns the style which is applied to the PLC code view.
- **SymbolLineBackColor**
  Specifies the background color of the symbol line.
- **SymbolLineFont**
  Specifies the font of the text in the symbol line.
- **SymbolLineForeColor**
  Specifies the text font color of the symbol line.
- **TabIndex**
  Returns the position of the PLC code view in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the PLC code view.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the PLC code view is visible.
- **Width**
  Specifies the width of the PLC code view.
- **WindowFlags**
  Specifies the window configuration of the PLC code view.

Methods

The "SystemDiagnosisControl" object has the following methods:

- **FireCommand()**
  Configures the occurrence of an event for an element.
- **Next()**
  Navigates to the next network.
- **OpenCodeViewerFromAlarm()**
  Opens the corresponding block in the PLC code view according to the selection in the alarm control.

- **OpenGRAPHDetails()**
  Opens the "GRAPH" details block in the PLC code view.
- **OpenGRAPHDetailsByConnectionName()**
  Opens the "GRAPH" details block in the PLC code view.
- **OpenProDiagDetailsFB()**
  Represents the logic of a network input of a function block in the PLC code view.
- **OpenProDiagDetailsFC()**
  Represents the logic of a function in the PLC code view, taking the UDT instance into account.
- **OpenProDiagDetailsNetwork()**
  Represents a network and its logic in the PLC code view.
- **Previous()**
  Navigates to the previous network.
- **ResetToConfiguration()**
  Executes the "ResetToConfiguration" command in the PLC code view.
- **ToggleCriteriaAnalysis()**
  Executes the "ToggleCriteriaAnalysis" command in the PLC code view.
- **ToggleGRAPHViewerMode()**
  Executes the "ToggleGRAPHViewerMode" command in the PLC code view.
- **ToggleNetworkDisplay()**
  Executes the "ToggleNetworkDisplay" command in the PLC code view.
- **ZoomIn()**
  Executes the "ZoomIn" command in the PLC code view.
- **ZoomOut()**
  Executes the "ZoomOut" command in the PLC code view.

Events

The "SystemDiagnosisControl" object has the following events:

- **OnActivated()**
  Occurs when a system diagnostics control receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the system diagnostics control.
- **OnDeactivated()**
  Occurs when a system diagnostics control loses focus.
- **OnInitialized()**
  Occurs when a system diagnostics control has been successfully initialized and the data connection to the PLC has been established.

ProcessDiagnosisPlcCodeViewerControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.BackColor
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Caption
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

ProcessDiagnosisPlcCodeViewerControl.Caption (Page 5283)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5284)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5284)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5284)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5284)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5284)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5284)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

ProcessDiagnosisPlcCodeViewerControl.Caption (Page 5283)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

ProcessDiagnosisPlcCodeViewerControl.Caption (Page 5283)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

ProcessDiagnosisPlcCodeViewerControl.Caption (Page 5283)

ProcessDiagnosisPlcCodeViewerControl.CaptionColor

Description

The "CaptionColor" property specifies the background color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.CaptionColor
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the PLC code view.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.CurrentQuality
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.CurrentZoomFactor

Description

The "CurrentZoomFactor" property specifies the zoom factor of the PLC code view. The zoom factor may differ from the containing screen. The value 1.0 corresponds to a zoom factor of 100%.

Type

Float

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.CurrentZoomFactor
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Enabled

Description

The "Enabled" property specifies whether the PLC code view can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Enabled
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Height

Description

The "Height" property specifies the height.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Height
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Icon

Description

The "Icon" property specifies the icon of the PLC code view.

Type

String

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Icon
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Layer

Description

The "Layer" property returns the screen layer in which the PLC code view is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Layer
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ProcessDiagnosisPlcCodeViewerControl.Layer (Page 5292)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ProcessDiagnosisPlcCodeViewerControl.Layer (Page 5292)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ProcessDiagnosisPlcCodeViewerControl.Layer (Page 5292)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ProcessDiagnosisPlcCodeViewerControl.Layer (Page 5292)

ProcessDiagnosisPlcCodeViewerControl.Left

Description

The "Left" property specifies the value of the X coordinate.

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Left
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.LineColors

Description

The "LineColors" property returns the line colors.

Type

Object, HmiLineColorPart

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.LineColors
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

LineColor.ActiveLine

Description

The "ActiveLine" property returns the active line.

Type

UInt32

Access

Read-only

Syntax

```
LineColor.ActiveLine
```

See also

ProcessDiagnosisPlcCodeViewerControl.LineColors (Page 5295)

LineColor.InactiveLine

Description

The "InactiveLine" property returns the inactive line.

Type

UInt32

Access

Read-only

Syntax

```
LineColor.InactiveLine
```

See also

ProcessDiagnosisPlcCodeViewerControl.LineColors (Page 5295)

LineColor.NormalLine

Description

The "NormalLine" property returns the normal line.

Type

UInt32

Access

Read-only

Syntax

```
LineColor.NormalLine
```

See also

ProcessDiagnosisPlcCodeViewerControl.LineColors (Page 5295)

ProcessDiagnosisPlcCodeViewerControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Margin
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ProcessDiagnosisPlcCodeViewerControl.Margin (Page 5297)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ProcessDiagnosisPlcCodeViewerControl.Margin (Page 5297)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ProcessDiagnosisPlcCodeViewerControl.Margin (Page 5297)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ProcessDiagnosisPlcCodeViewerControl.Margin (Page 5297)

ProcessDiagnosisPlcCodeViewerControl.Name

Description

The "Name" property returns the name of the PLC code view.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Name
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OverviewDetailRatio

Description

The "OverviewDetailRatio" property specifies how much space the detail view takes up in the PLC code view.

Type

Float

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OverviewDetailRatio
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 5301)

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Parent
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

Screen Items (Page 5301)

Screen Items

Description

Screen Items (Page 1571)

ProcessDiagnosisPlcCodeViewerControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the PLC code view was created.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.RenderingTemplate
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the PLC code view is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ShowFocusVisual
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ShowSymbolLine

Description

The "ShowSymbolLine" property specifies whether the symbol line is displayed.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ShowSymbolLine
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.StatusBar

Description

The "StatusBar" property specifies the information bar.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.StatusBar
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 5304)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

HmiControlBarElementCollection (Page 5304)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 5304)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 5304)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 5306)

See also

HmiControlBarElementCollection (Page 5304)

Control Bar Elements (Page 5306)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5304)

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 5306)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 5306)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 5306)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 5306)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 5306)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 5312)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 5312)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 5312)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 5312)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 5312)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 5312)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 5312)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 5312)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 5306)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 5306)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 5306)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 5322)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 5322)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 5322)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 5322)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 5306)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 5306)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 5306)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 5306)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 5327)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 5327)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 5327)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 5327)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 5306)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 5306)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 5306)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5304)

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 5332)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 5334)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 5334)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 5334)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 5334)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 5334)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 5334)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 5334)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 5334)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment

- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms

- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 5332)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 5344)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 5344)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 5344)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 5344)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 5332)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 5349)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 5349)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 5349)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 5349)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 5332)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 5332)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5304)

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 5354)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 5361)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 5361)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 5361)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 5361)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 5354)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 5366)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 5366)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 5366)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 5366)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 5354)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 5354)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5304)

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line

- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time

- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 5372)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 5378)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 5378)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 5378)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 5378)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 5372)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 5383)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 5383)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 5383)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 5383)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 5372)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 5372)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.

- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5304)

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 5388)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 5397)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 5397)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 5397)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 5397)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 5388)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 5402)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 5402)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 5402)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 5402)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 5388)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 5388)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.

- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.

- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5304)

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 5408)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 5415)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5415)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 5408)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 5426)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 5426)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 5426)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 5426)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 5408)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 5431)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 5431)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 5431)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 5431)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 5408)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 5408)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 5437)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 5437)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 5437)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 5437)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 5437)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 5437)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 5440)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 5440)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 5440)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 5440)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 5443)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 5443)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 5443)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 5443)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

ProcessDiagnosisPlcCodeViewerControl.StatusBar (Page 5303)

ProcessDiagnosisPlcCodeViewerControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the PLC code view.

Type

String

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.StyleItemClass
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.SymbolLineBackColor

Description

The "SymbolLineBackColor" property specifies the background color of the symbol line.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.SymbolLineBackColor
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont

Description

The "SymbolLineFont" property specifies the font of the text in the symbol line.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont (Page 5447)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont (Page 5447)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont (Page 5447)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont (Page 5447)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont (Page 5447)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ProcessDiagnosisPlcCodeViewerControl.SymbolLineFont (Page 5447)

ProcessDiagnosisPlcCodeViewerControl.SymbolLineForeColor

Description

The "SymbolLineForeColor" property specifies the text font color of the symbol line.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.SymbolLineForeColor
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.TabIndex

Description

The "TabIndex" property returns the position of the PLC code view in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.TabIndex
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the PLC code view.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ToolBar
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 5454)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

HmiControlBarElementCollection (Page 5454)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 5453)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 5454)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 5456)

See also

HmiControlBarElementCollection (Page 5454)

Control Bar Elements (Page 5456)

Control Bar Elements

Description

Control Bar Elements (Page 5306)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 5456)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 5456)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 5456)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 5456)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 5456)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 5456)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 5460)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 5460)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 5460)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 5460)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 5462)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 5462)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 5462)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 5462)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

ProcessDiagnosisPlcCodeViewerControl.ToolBar (Page 5452)

ProcessDiagnosisPlcCodeViewerControl.Top

Description

The "Top" property specifies the value of the Y coordinate.

Type

Int32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Top
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Visible

Description

The "Visible" property specifies whether the PLC code view is visible.

Type

Bool

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Visible
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Width

Description

The "Width" property specifies the width.

Type

UInt32

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Width
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the system PLC code view.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.WindowFlags
```

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Next()

Description

The "Next" method navigates to the next network.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Next()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OpenCodeViewerFromAlarm()

Description

The "OpenCodeViewerFromAlarm" method opens the corresponding block in the PLC code view according to the selection in the alarm control.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OpenCodeViewerFromAlarm(HmiConn
ectionName, CpuAlarmID, TextlistIndex)
```

Parameters

HmiConnectionName

Type: String

Connection name

CpuAlarmID

Type: Object, HMIUint64

ID of the CPU alarm

TextlistIndex

Type: Object, HMIUint64

Index of the text list

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OpenGRAPHDetails()

Description

The "OpenGRAPHDetails" method opens the "GRAPH" details block in the PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OpenGRAPHDetails(PlcName,
Block, StepNumber)
```

Parameters

PlcName

Type: String

Name of the PLC

Block

Type: String

Name of the block

StepNumber

Type: UInt16

Step number

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OpenGRAPHDetailsByConnectionName()

Description

The "OpenGRAPHDetailsByConnectionName" method opens the "GRAPH" details block in the PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OpenGRAPHDetailsByConnectionNam
e(HmiConnectionName, Block, StepNumber)
```

Parameters

HmiConnectionName

Type: String

Connection name

Block

Type: String

Name of the block

StepNumber

Type: UInt16

Step number

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OpenProDiagDetailsFB()

Description

The "OpenProDiagDetailsFB" method represents the logic of a network input of a function block in the PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OpenProDiagDetailsFB(PlcName,
ContainingBlock, CallBlock, Pin, PinSubstringSearch)
```

Parameters

PlcName

Type: String

Name of the PLC.

ContainingBlock

Type: String

Name of the contained block

`CallBlock`

Type: String

Name of the called block

`Pin`

Type: String

Name of the input pin of the "CallBlock".

`PinSubstringSearch`

Type: Bool

Specifies whether the pin name starts with the transferred pin parameter.

- True: Pin name starts with the transferred pin parameter.
- False: Pin name must be the same as the pin parameter.

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OpenProDiagDetailsFC()

Description

The "OpenProDiagDetailsFC" represents the logic of a function in the PLC code view, taking the UDT instance into account.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OpenProDiagDetailsFC(PlcName,
ContainingBlock, CallBlock, Pin, UdtInstance, PinSubstringSearch)
```

Parameters

PlcName

Type: String

Name of the PLC.

ContainingBlock

Type: String

Name of the contained block

CallBlock

Type: String

Name of the called block

Pin

Type: String

Name of the input pin of the "CallBlock".

UdtInstance

Type: String

UDT instance that is used to limit the display of FCs called multiple times.

PinSubstringSearch

Type: Bool

Specifies whether the pin name starts with the transferred pin parameter.

- True: Pin name starts with the transferred pin parameter
- False: Pin name must be the same as the pin parameter

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.OpenProDiagDetailsNetwork()

Description

The "OpenProDiagDetailsNetwork" method represents a network and its logic in the PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.OpenProDiagDetailsNetwork(PlcName, ContainingBlock, Operand)
```

Parameters

PlcName

Type: String

Name of the PLC

ContainingBlock

Type: String

Name of the contained block

Operand

Type: String

Operand of the access point

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.Previous()

Description

The "Previous" method navigates to the previous network.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.Previous()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ResetToConfiguration()

Description

The "ResetToConfiguration" method executes the "ResetToConfiguration" command in der PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ResetToConfiguration()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ToggleCriteriaAnalysis()

Description

The "ToggleCriteriaAnalysis" method executes the "ToggleCriteriaAnalysis" command in der PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ToggleCriteriaAnalysis()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ToggleGRAPHViewerMode()

Description

The "ToggleGRAPHViewerMode" method executes the "ToggleGRAPHViewerMode" command in der PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ToggleGRAPHViewerMode()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ToggleNetworkDisplay()

Description

The "ToggleNetworkDisplay" method executes the "ToggleNetworkDisplay" command in der PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ToggleNetworkDisplay()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ZoomIn()

Description

The "ZoomIn" method executes the "ZoomIn" command in der PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ZoomIn()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

ProcessDiagnosisPlcCodeViewerControl.ZoomOut()

Description

The "ZoomOut" method executes the "ZoomOut" command in der PLC code view.

Syntax

```
ProcessDiagnosisPlcCodeViewerControl.ZoomOut()
```

Parameters

--

Return value

--

See also

ProcessDiagnosisPlcCodeViewerControl (Page 5280)

RadioButtonGroup

Description

The "RadioButtonGroup" object represents an option button in runtime.

Object type

HmiRadioButtonGroup

Properties

The "RadioButtonGroup" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **Content**
  Specifies the display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the option button.
- **Enabled**
  Specifies whether the option button can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the option button is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the option button.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the option button is operable.
- **Padding**
  Specifies the distance of the content from the border of the option button.
- **Parent**
  Returns the higher-level screen object.

- **ProcessValue**
  Specifies the process value.
- **RenderingTemplate**
  Returns the name of the template from which the option button was created.
- **RequireExplicitUnlock**
  Returns whether the option button is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the option button rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **SelectionItemHeight**
  Specifies the height of the option button entries.
- **SelectionItems**
  Returns the list of all option button entries.
- **SelectorPosition**
  Specifies the horizontal alignment of the option button entries.
- **ShowFocusVisual**
  Specifies whether the option button is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the option button.
- **TabIndex**
  Returns the position of the option button in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the option button is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "RadioButtonGroup" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the option button.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "RadioButtonGroup" object has the following events:

- **OnActivated()**
  Occurs when an option button receives focus.
- **OnContextTapped()**
  Occurs when an option button is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when an option button loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the option button is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the option button is in focus.
- **OnTapped()**
  Occurs when an option button is left-clicked or short-touched.

RadioButtonGroup.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.AlternateBackColor
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.AlternateBorderColor
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
RadioButtonGroup.Authorization
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.BackColor
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.BorderColor
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
RadioButtonGroup.BorderWidth
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
RadioButtonGroup.Content
```

See also

RadioButtonGroup (Page 5478)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

RadioButtonGroup.Content (Page 5484)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

RadioButtonGroup.Content (Page 5484)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

RadioButtonGroup.Content (Page 5484)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

RadioButtonGroup.Content (Page 5484)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

RadioButtonGroup.Content (Page 5484)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

RadioButtonGroup.Content (Page 5484)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

RadioButtonGroup.Content (Page 5484)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

RadioButtonGroup.Content (Page 5484)

RadioButtonGroup.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the option button.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
RadioButtonGroup.CurrentQuality
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Enabled

Description

The "Enabled" property specifies whether the option button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
RadioButtonGroup.Enabled
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
RadioButtonGroup.Font
```

See also

RadioButtonGroup (Page 5478)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

RadioButtonGroup.Font (Page 5491)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

RadioButtonGroup.Font (Page 5491)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

RadioButtonGroup.Font (Page 5491)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

RadioButtonGroup.Font (Page 5491)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

RadioButtonGroup.Font (Page 5491)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

RadioButtonGroup.Font (Page 5491)

RadioButtonGroup.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.ForeColor
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.Height
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Layer

Description

The "Layer" property returns the layer of the screen in which the option button is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
RadioButtonGroup.Layer
```

See also

RadioButtonGroup (Page 5478)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

RadioButtonGroup.Layer (Page 5495)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

RadioButtonGroup.Layer (Page 5495)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

RadioButtonGroup.Layer (Page 5495)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

RadioButtonGroup.Layer (Page 5495)

RadioButtonGroup.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
RadioButtonGroup.Left
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
RadioButtonGroup.Margin
```

See also

RadioButtonGroup (Page 5478)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

RadioButtonGroup.Margin (Page 5498)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

RadioButtonGroup.Margin (Page 5498)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

RadioButtonGroup.Margin (Page 5498)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

RadioButtonGroup.Margin (Page 5498)

RadioButtonGroup.Name

Description

The "Name" property returns the name of the option button.

Type

String

Access

Read-only

Syntax

```
RadioButtonGroup.Name
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
RadioButtonGroup.Opacity
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Operability

Description

The "Operability" property returns whether the option button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
RadioButtonGroup.Operability
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Padding

Description

The "Padding" property specifies the distance of the content from the border of the option button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
RadioButtonGroup.Padding
```

See also

RadioButtonGroup (Page 5478)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

RadioButtonGroup.Padding (Page 5502)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

RadioButtonGroup.Padding (Page 5502)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

RadioButtonGroup.Padding (Page 5502)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

RadioButtonGroup.Padding (Page 5502)

RadioButtonGroup.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
RadioButtonGroup.Parent
```

See also

RadioButtonGroup (Page 5478)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

RadioButtonGroup.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
RadioButtonGroup.ProcessValue
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the option button was created.

Type

String

Access

Read-only

Syntax

```
RadioButtonGroup.RenderingTemplate
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the option button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
RadioButtonGroup.RequireExplicitUnlock
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
RadioButtonGroup.RotationAngle
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RotationCenterPlacement (Page 5508)

RadioButtonGroup.RotationCenterX (Page 5508)

RadioButtonGroup.RotationCenterY (Page 5509)

RadioButtonGroup.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the option button rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
RadioButtonGroup.RotationCenterPlacement
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RotationAngle (Page 5507)

RadioButtonGroup.RotationCenterX (Page 5508)

RadioButtonGroup.RotationCenterY (Page 5509)

RadioButtonGroup.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
RadioButtonGroup.RotationCenterX
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RotationAngle (Page 5507)

RadioButtonGroup.RotationCenterPlacement (Page 5508)

RadioButtonGroup.RotationCenterY (Page 5509)

RadioButtonGroup.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
RadioButtonGroup.RotationCenterY
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.RotationAngle (Page 5507)

RadioButtonGroup.RotationCenterPlacement (Page 5508)

RadioButtonGroup.RotationCenterX (Page 5508)

RadioButtonGroup.SelectionItemHeight

Description

The "SelectionItemHeight" property specifies the height of the option button entries. The value "0" indicates that the height is calculated automatically.

Type

UInt16

Access

Read-write

Syntax

```
RadioButtonGroup.SelectionItemHeight
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.SelectionItems

Description

The "SelectionItems" property returns the list of all option button entries ("SelectionItem" objects).

Type

Object, HmiSelectionItemCollection (Page 5511)

Access

Read-only

Syntax

```
RadioButtonGroup.SelectionItems
```

See also

RadioButtonGroup (Page 5478)

HmiSelectionItemCollection (Page 5511)

HmiSelectionItemCollection

Description

The "HmiSelectionItemCollection" object is a list of all entries ("SelectionItem" objects) of a list object.

Use

The "HmiSelectionItemCollection" object is a list and can be counted and enumerated. You can access the "HmiSelectionItemCollection" list using the index or the tag name.

Object type

HmiSelectionItemCollection

Properties

The "HmiSelectionItemCollection" object has the following properties:

- **Count**
  Returns the number of list entries of the "HmiSelectionItemCollection" list.

Methods

The "HmiSelectionItemCollection" object has the following methods:

- **Item()**
  Returns a list entry of the "HmiSelectionItemCollection" list.

See also

RadioButtonGroup.SelectionItems (Page 5510)

HmiSelectionItemCollection.Count

Description

The "Count" property returns the number of list entries in the "HmiSelectionItemCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiSelectionItemCollection.Count
```

See also

HmiSelectionItemCollection (Page 5511)

HmiSelectionItemCollection.Item()

Description

The "Item" method returns a list entry of the "HmiSelectionItemCollection" list.

Syntax

```
HmiSelectionItemCollection[.Item](HmiSelectionItemName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiSelectionItemCollection" object.

Parameters

`HmiSelectionItemName`

Type: String

Name of the list entry

Return value

Object, HmiSelectionItemPart (Page 5512)

See also

HmiSelectionItemCollection (Page 5511)

SelectionItem (Page 5512)

SelectionItem

Description

The "SelectionItem" object represents a list entry.

Object type

HmiSelectionItemPart

Properties

The "SelectionItem" object has the following properties:

- **Graphic**
  Specifies the graphic of the list entry.
- **IsSelected**
  Specifies whether the list entry is selected.
- **Text**
  Specifies the list entry text.

Methods

--

SelectionItem.Graphic

Description

The "Graphic" property specifies the graphic of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Graphic
```

See also

SelectionItem (Page 5512)

SelectionItem.IsSelected

Description

The "IsSelected" property specifies whether the list entry is selected.

Type

Bool

Access

Read-write

Syntax

```
SelectionItem.IsSelected
```

See also

SelectionItem (Page 5512)

SelectionItem.Text

Description

The "Text" property specifies the text of the list entry.

Type

String

Access

Read-write

Syntax

```
SelectionItem.Text
```

See also

SelectionItem (Page 5512)

RadioButtonGroup.SelectorPosition

Description

The "SelectorPosition" property specifies the horizontal alignment of the option button entries.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
RadioButtonGroup.SelectorPosition
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the option button is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
RadioButtonGroup.ShowFocusVisual
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the option button.

Type

String

Access

Read-only

Syntax

```
RadioButtonGroup.StyleItemClass
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.TabIndex

Description

The "TabIndex" property returns the position of the option button in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
RadioButtonGroup.TabIndex
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
RadioButtonGroup.ToolTipText
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
RadioButtonGroup.Top
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Visible

Description

The "Visible" property specifies whether the option button is visible.

Type

Bool

Access

Read-write

Syntax

```
RadioButtonGroup.Visible
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
RadioButtonGroup.VisualizeQuality
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
RadioButtonGroup.Width
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the option button.

Syntax

```
RadioButtonGroup.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
RadioButtonGroup.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup_OnActivated()

Description

The "OnActivated" event occurs when an option button receives focus:

- An option button is selected via the configured tab sequence.
- An option button that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
RadioButtonGroup_OnActivated(item)
```

Context

item

Type: Object

Option button where the event occurs.

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- An option button is right-clicked.
- An option button is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
RadioButtonGroup_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Option button where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup_OnDeactivated()

Description

The "OnDeactivated" event occurs when the option button loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
RadioButtonGroup_OnDeactivated(item)
```

Context

`item`

Type: Object

Option button where the event occurs.

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the option button is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
RadioButtonGroup_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Option button where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the option button is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
RadioButtonGroup_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Option button where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

RadioButtonGroup (Page 5478)

RadioButtonGroup_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- An option button is left-clicked.
- The <RETURN> or <SPACE> key is pressed when an option button has the focus.
- An option button is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
RadioButtonGroup_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Option button where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

RadioButtonGroup (Page 5478)

Rectangle

Description

The "Rectangle" object represents a rectangle in runtime.

Object type

HmiRectangle

Properties

The "Rectangle" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **Corners**
  Specifies the rounding of the rectangle corners.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the rectangle.
- **DashType**
  Specifies the stroke style of the border or line.
- **Enabled**
  Specifies whether the rectangle can be operated in runtime.
- **FillDirection**
  Specifies the direction from which the rectangle is filled.
- **FillLevel**
  Specifies the fill of the rectangle in percent.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the rectangle is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the rectangle.

- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the rectangle is operable.
- **Parent**
  Returns the higher-level screen object.
- **RequireExplicitUnlock**
  Returns whether the rectangle is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the rectangle rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFillLevel**
  Specifies whether the fill level is displayed.
- **ShowFocusVisual**
  Specifies whether the rectangle is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the rectangle.
- **TabIndex**
  Returns the position of the rectangle in the tab sequence.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the rectangle is visible.
- **Width**
  Specifies the width.

Methods

The "Rectangle" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the rectangle.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Rectangle" object has the following events:

- **OnActivated()**
  Occurs when a rectangle receives focus.
- **OnContextTapped()**
  Occurs when a rectangle is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a rectangle loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the rectangle is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the rectangle is in focus.
- **OnTapped()**
  Occurs when a rectangle is left-clicked or short-touched.

Rectangle.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Rectangle.AlternateBackColor
```

See also

Rectangle (Page 5527)

Rectangle.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Rectangle.AlternateBorderColor
```

See also

Rectangle (Page 5527)

Rectangle.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Rectangle.Authorization
```

See also

Rectangle (Page 5527)

Rectangle.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Rectangle.BackColor
```

See also

Rectangle (Page 5527)

Rectangle.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient
- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
Rectangle.BackFillPattern
```

See also

Rectangle (Page 5527)

Rectangle.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Rectangle.BorderColor
```

See also

Rectangle (Page 5527)

Rectangle.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Rectangle.BorderWidth
```

See also

Rectangle (Page 5527)

Rectangle.Corners

Description

The "Corners" property specifies the rounding of the rectangle corners.

Type

Object, HmiCornersPart

Access

Read-write

Syntax

```
Rectangle.Corners
```

See also

Rectangle (Page 5527)

Corners.BottomLeftRadius

Description

The "BottomLeftRadius" property specifies the radius of the rounding of the bottom-left corner.

Type

UInt32

Access

Read-write

Syntax

```
Corners.BottomLeftRadius
```

See also

Rectangle.Corners (Page 5534)

Corners.BottomRightRadius

Description

The "BottomRightRadius" property specifies the radius of the rounding of the bottom-right corner.

Type

UInt32

Access

Read-write

Syntax

```
Corners.BottomRightRadius
```

See also

Rectangle.Corners (Page 5534)

Corners.TopLeftRadius

Description

The "TopLeftRadius" property specifies the radius of the rounding of the top-left corner.

Type

UInt32

Access

Read-write

Syntax

```
Corners.TopLeftRadius
```

See also

Rectangle.Corners (Page 5534)

Corners.TopRightRadius

Description

The "TopRightRadius" property specifies the radius of the rounding of the top-right corner.

Type

UInt32

Access

Read-write

Syntax

```
Corners.TopRightRadius
```

See also

Rectangle.Corners (Page 5534)

Rectangle.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the rectangle.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Rectangle.CurrentQuality
```

See also

Rectangle (Page 5527)

Rectangle.DashType

Description

The "DashType" property specifies the stroke type of the border or line.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dotted
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Rectangle.DashType
```

See also

Rectangle (Page 5527)

Rectangle.Enabled

Description

The "Enabled" property specifies whether the rectangle can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Rectangle.Enabled
```

See also

Rectangle (Page 5527)

Rectangle.FillDirection

Description

The "FillDirection" property specifies the direction from which the rectangle is filled.

Type

Int32, HmiFillDirection

Specifies the filling direction:

- BottomToTop (0): From bottom to top
- TopToBottom (1): From top to bottom
- LeftToRight (2): From left to right
- RightToLeft (3): From right to left

Access

Read-write

Syntax

```
Rectangle.FillDirection
```

See also

Rectangle (Page 5527)

Rectangle.FillLevel

Description

The "FillLevel" property specifies the fill level of the rectangle in percent.

Type

UInt8

Access

Read-write

Syntax

```
Rectangle.FillLevel
```

See also

Rectangle (Page 5527)

Rectangle.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Rectangle.Height
```

See also

Rectangle (Page 5527)

Rectangle.Layer

Description

The "Layer" property returns the layer of the screen in which the rectangle is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Rectangle.Layer
```

See also

Rectangle (Page 5527)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Rectangle.Layer (Page 5540)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Rectangle.Layer (Page 5540)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Rectangle.Layer (Page 5540)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Rectangle.Layer (Page 5540)

Rectangle.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Rectangle.Left
```

See also

Rectangle (Page 5527)

Rectangle.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Rectangle.Margin
```

See also

Rectangle (Page 5527)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Rectangle.Margin (Page 5543)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Rectangle.Margin (Page 5543)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Rectangle.Margin (Page 5543)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Rectangle.Margin (Page 5543)

Rectangle.Name

Description

The "Name" property returns the name of the rectangle.

Type

String

Access

Read-only

Syntax

```
Rectangle.Name
```

See also

Rectangle (Page 5527)

Rectangle.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Rectangle.Opacity
```

See also

Rectangle (Page 5527)

Rectangle.Operability

Description

The "Operability" property returns whether the rectangle is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Rectangle.Operability
```

See also

Rectangle (Page 5527)

Rectangle.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Rectangle.Parent
```

See also

Rectangle (Page 5527)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Rectangle.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the rectangle can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Rectangle.RequireExplicitUnlock
```

See also

Rectangle (Page 5527)

Rectangle.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Rectangle.RotationAngle
```

See also

Rectangle (Page 5527)

Rectangle.RotationCenterPlacement (Page 5548)

Rectangle.RotationCenterX (Page 5549)

Rectangle.RotationCenterY (Page 5550)

Rectangle.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the rectangle rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Rectangle.RotationCenterPlacement
```

See also

Rectangle (Page 5527)

Rectangle.RotationAngle (Page 5548)

Rectangle.RotationCenterX (Page 5549)

Rectangle.RotationCenterY (Page 5550)

Rectangle.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Rectangle.RotationCenterX
```

See also

Rectangle (Page 5527)

Rectangle.RotationAngle (Page 5548)

Rectangle.RotationCenterPlacement (Page 5548)

Rectangle.RotationCenterY (Page 5550)

Rectangle.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Rectangle.RotationCenterY
```

See also

Rectangle (Page 5527)

Rectangle.RotationAngle (Page 5548)

Rectangle.RotationCenterPlacement (Page 5548)

Rectangle.RotationCenterX (Page 5549)

Rectangle.ShowFillLevel

Description

The "ShowFillLevel" property specifies whether the fill level is displayed.

Type

Bool

Access

Read-write

Syntax

```
Rectangle.ShowFillLevel
```

See also

Rectangle (Page 5527)

Rectangle.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the rectangle is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Rectangle.ShowFocusVisual
```

See also

Rectangle (Page 5527)

Rectangle.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the rectangle.

Type

String

Access

Read-only

Syntax

```
Rectangle.StyleItemClass
```

See also

Rectangle (Page 5527)

Rectangle.TabIndex

Description

The "TabIndex" property returns the position of the rectangle in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Rectangle.TabIndex
```

See also

Rectangle (Page 5527)

Rectangle.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Rectangle.ToolTipText
```

See also

Rectangle (Page 5527)

Rectangle.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Rectangle.Top
```

See also

Rectangle (Page 5527)

Rectangle.Visible

Description

The "Visible" property specifies whether the rectangle is visible.

Type

Bool

Access

Read-write

Syntax

```
Rectangle.Visible
```

See also

Rectangle (Page 5527)

Rectangle.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Rectangle.Width
```

See also

Rectangle (Page 5527)

Rectangle.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the rectangle.

Syntax

```
Rectangle.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Rectangle (Page 5527)

Rectangle.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Rectangle.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Rectangle (Page 5527)

Rectangle_OnActivated()

Description

The "OnActivated" event occurs when a rectangle receives focus:

- A rectangle is selected via the configured tab sequence.
- A rectangle that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Rectangle_OnActivated(item)
```

Context

item

Type: Object

Rectangle where the event occurs.

See also

Rectangle (Page 5527)

Rectangle_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A rectangle is right-clicked.
- A rectangle is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Rectangle_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Rectangle where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Rectangle (Page 5527)

Rectangle_OnDeactivated()

Description

The "OnDeactivated" event occurs when the rectangle loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Rectangle_OnDeactivated(item)
```

Context

`item`

Type: Object

Rectangle where the event occurs.

See also

Rectangle (Page 5527)

Rectangle_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the rectangle is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Rectangle_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Rectangle where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Rectangle (Page 5527)

Rectangle_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the rectangle is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Rectangle_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Rectangle where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Rectangle (Page 5527)

Rectangle_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A rectangle is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a rectangle has the focus.
- A rectangle is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Rectangle_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Rectangle where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Rectangle (Page 5527)

ScreenWindow

Description

ScreenWindow (Page 1436)

Slider

Description

The "Slider" object represents a slider for monitoring of process values in runtime.

Object type

HmiSlider

Properties

The "Slider" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BarMode**
  Specifies the color display of the slider.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **ComputedMaxPeakValue**
  Returns the highest process value that occurred.
- **ComputedMinPeakValue**
  Returns the lowest process value that occurred.
- **ComputedValueTendency**
  Returns the change of the process value.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the slider.
- **Enabled**
  Specifies whether the slider can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **Height**
  Specifies the height.
- **Label**
  Specifies the labeling below the slider.

- **Layer**
  Returns the layer of the screen in which the slider is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the slider.
- **NormalRangeColor**
  Specifies the color of the normal range.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the slider is operable.
- **OriginValue**
  Specifies the output value of the normal range that is visualized.
- **OutputFormat**
  Specifies the format for displaying the process values.
- **Parent**
  Returns the higher-level screen object.
- **PeakIndicators**
  Specifies whether the highest and lowest process value up to this time are displayed.
- **ProcessValue**
  Specifies the process value.
- **ProcessValueIndicatorBackColor**
  Specifies the background color for the process value.
- **ProcessValueIndicatorForeColor**
  Specifies the foreground color for the process value.
- **ProcessValueIndicatorMode**
  Specifies the type of display of the current process value.
- **RelativeToOrigin**
  Specifies whether the output value is an absolute or a percentage value between the minimum and maximum value.
- **RenderingTemplate**
  Returns the name of the template from which the slider was created.
- **RequireExplicitUnlock**
  Returns whether the slider is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the slider rotates.

- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ScaleBackColor**
  Specifies the background color of the scale.
- **ScaleForeColor**
  Specifies the foreground color of the scale.
- **ShowFocusVisual**
  Specifies whether the slider is highlighted when in focus.
- **ShowTrendIndicator**
  Specifies whether the tendency (rising or falling) of the process value to be monitored is indicated by means of a small arrow.
- **ShowValue**
  Specifies whether the process value is additionally output as text.
- **StraightScale**
  Specifies the scale of the slider.
- **StyleItemClass**
  Returns the style which is applied to the slider.
- **TabIndex**
  Returns the position of the slider in the tab sequence.
- **ThresholdIndicators**
  Specifies how parameterized limit values are visualized.
- **Thresholds**
  Returns the list of all limit values of the slider.
- **ThumbBackColor**
  Specifies the background color of the slider.
- **ThumbForeColor**
  Specifies the foreground color of the slider.
- **Title**
  Specifies the caption which appears as the title.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **TrendIndicatorColor**
  Specifies the color of the trend indicator.
- **ValuePosition**
  Specifies where the value of the current slider position is additionally displayed numerically.
- **Visible**
  Specifies whether the slider is visible.

- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.
- **WriteDuringChange**
  Specifies when changes are transferred.

Methods

The "Slider" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the slider.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Slider" object has the following events:

- **OnActivated()**
  Occurs when a slider receives focus.
- **OnContextTapped()**
  Occurs when a slider is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a slider loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the slider is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the slider is in focus.
- **OnTapped()**
  Occurs when a slider is left-clicked or short-touched.

Slider.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Slider.AlternateBackColor
```

See also

Slider (Page 5563)

Slider.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
Slider.AlternateBorderColor
```

See also

Slider (Page 5563)

Slider.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Slider.Authorization
```

See also

Slider (Page 5563)

Slider.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Slider.BackColor
```

See also

Slider (Page 5563)

Slider.BarMode

Description

The "BarMode" property specifies the color of the slider.

Type

Int32, HmiBarMode

Specifies the bar mode:

- Segmented (0): Bar changes color according to the bar segments.
- Unicolor (1): Entire bar has same color.

- SegmentedStatic (2): Bar segments in the background, process value indicator in front of the bar segments.
- UnicolorStatic (3): Background color changes according to the process value and the limit colors, process value indicator runs in front of the bar segments.

Access

Read-write

Syntax

```
Slider.BarMode
```

See also

Slider (Page 5563)

Slider.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
Slider.BorderColor
```

See also

Slider (Page 5563)

Slider.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
Slider.BorderWidth
```

See also

Slider (Page 5563)

Slider.ComputedMaxPeakValue

Description

The "ComputedMaxPeakValue" property returns the highest process value that occurred.

Type

Variant

Access

Read-only

Syntax

```
Slider.ComputedMaxPeakValue
```

See also

Slider (Page 5563)

Slider.ComputedMinPeakValue (Page 5570)

Slider.ComputedMinPeakValue

Description

The "ComputedMinPeakValue" property returns the lowest process value which occurred.

Type

Variant

Access

Read-only

Syntax

```
Slider.ComputedMinPeakValue
```

See also

Slider (Page 5563)

Slider.ComputedMaxPeakValue (Page 5570)

Slider.ComputedValueTendency

Description

The "ComputedValueTendency" property returns the change in the process value.

Type

Int32, HmiValueTendency

Returns the modification:

- Steady (0): No change
- Upwards (1): Change upwards
- Downwards (2): Change downwards

Access

Read-only

Syntax

```
Slider.ComputedValueTendency
```

See also

Slider (Page 5563)

Slider.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the slider.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Slider.CurrentQuality
```

See also

Slider (Page 5563)

Slider.Enabled

Description

The "Enabled" property specifies whether the slider can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Slider.Enabled
```

See also

Slider (Page 5563)

Slider.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Slider.Font
```

See also

Slider (Page 5563)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Slider.Font (Page 5573)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Slider.Font (Page 5573)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Slider.Font (Page 5573)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Slider.Font (Page 5573)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Slider.Font (Page 5573)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Slider.Font (Page 5573)

Slider.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Slider.Height
```

See also

Slider (Page 5563)

Slider.Label

Description

The "Label" property specifies the labeling below the slider.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Slider.Label
```

See also

Slider (Page 5563)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Slider.Label (Page 5577)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5577)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5577)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5577)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5577)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5577)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5577)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Slider.Label (Page 5577)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Slider.Label (Page 5577)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Slider.Label (Page 5577)

Slider.Layer

Description

The "Layer" property returns the layer of the screen in which the slider is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Slider.Layer
```

See also

Slider (Page 5563)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Slider.Layer (Page 5583)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

Slider.Layer (Page 5583)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Slider.Layer (Page 5583)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Slider.Layer (Page 5583)

Slider.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Slider.Left
```

See also

Slider (Page 5563)

Slider.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Slider.Margin
```

See also

Slider (Page 5563)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Slider.Margin (Page 5586)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Slider.Margin (Page 5586)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Slider.Margin (Page 5586)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Slider.Margin (Page 5586)

Slider.Name

Description

The "Name" property returns the name of the slider.

Type

String

Access

Read-only

Syntax

```
Slider.Name
```

See also

Slider (Page 5563)

Slider.NormalRangeColor

Description

The "NormalRangeColor" property specifies the color of the normal range.

Type

UInt32

Access

Read-write

Syntax

```
Slider.NormalRangeColor
```

See also

Slider (Page 5563)

Slider.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
Slider.Opacity
```

See also

Slider (Page 5563)

Slider.Operability

Description

The "Operability" property returns whether the slider is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Slider.Operability
```

See also

Slider (Page 5563)

Slider.OriginValue

Description

The "OriginValue" property specifies the output value of the normal range to be visualized.

Type

Float

Access

Read-write

Syntax

```
Slider.OriginValue
```

See also

Slider (Page 5563)

Slider.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
Slider.OutputFormat
```

See also

Slider (Page 5563)

Slider.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Slider.Parent
```

See also

Slider (Page 5563)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Slider.PeakIndicators

Description

The "PeakIndicators" property specifies whether the highest and lowest process value up to this time are displayed.

Type

Int32, HmiPeakIndicator

Specifies the display of the peak indicator:

- None (0): No display
- Low (1): Only the lowest process value
- High (2): Only the highest process value

Access

Read-write

Syntax

```
Slider.PeakIndicators
```

See also

Slider (Page 5563)

Slider.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
Slider.ProcessValue
```

See also

Slider (Page 5563)

Slider.ProcessValueIndicatorBackColor

Description

The "ProcessValueIndicatorBackColor" property specifies the background color for the process value indicator.

Type

UInt32

Access

Read-write

Syntax

```
Slider.ProcessValueIndicatorBackColor
```

See also

Slider (Page 5563)

Slider.ProcessValueIndicatorForeColor (Page 5593)

Slider.ProcessValueIndicatorForeColor

Description

The "ProcessValueIndicatorForeColor" property specifies the foreground color for the process value indicator.

Type

UInt32

Access

Read-write

Syntax

```
Slider.ProcessValueIndicatorForeColor
```

See also

Slider (Page 5563)

Slider.ProcessValueIndicatorBackColor (Page 5593)

Slider.ProcessValueIndicatorMode

Description

The "ProcessValueIndicatorMode" property specifies the type of display of the current process value.

Type

Int32, HmiProcessIndicatorMode

Specifies the type of display:

- Bar (0): Bar only
- Indicator (1): Hair line or needle, no numerical display of the process value.
- DetailedIndicator (2): Detailed display with numerical value
- BarWithDetailedIndicator (3): Bar with numerical value

Access

Read-write

Syntax

```
Slider.ProcessValueIndicatorMode
```

See also

Slider (Page 5563)

Slider.RelativeToOrigin

Description

The "RelativeToOrigin" property specifies whether the output value is an absolute or a percentage value between minimum and maximum value.

Type

Bool

Access

Read-write

Syntax

```
Slider.RelativeToOrigin
```

See also

Slider (Page 5563)

Slider.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the slider was created.

Type

String

Access

Read-only

Syntax

```
Slider.RenderingTemplate
```

See also

Slider (Page 5563)

Slider.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the slider can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Slider.RequireExplicitUnlock
```

See also

Slider (Page 5563)

Slider.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
Slider.RotationAngle
```

See also

Slider (Page 5563)

Slider.RotationCenterPlacement (Page 5597)

Slider.RotationCenterX (Page 5597)

Slider.RotationCenterY (Page 5598)

Slider.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the slider rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Slider.RotationCenterPlacement
```

See also

Slider (Page 5563)

Slider.RotationAngle (Page 5596)

Slider.RotationCenterX (Page 5597)

Slider.RotationCenterY (Page 5598)

Slider.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Slider.RotationCenterX
```

See also

Slider (Page 5563)

Slider.RotationAngle (Page 5596)

Slider.RotationCenterPlacement (Page 5597)

Slider.RotationCenterY (Page 5598)

Slider.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
Slider.RotationCenterY
```

See also

Slider (Page 5563)

Slider.RotationAngle (Page 5596)

Slider.RotationCenterPlacement (Page 5597)

Slider.RotationCenterX (Page 5597)

Slider.ScaleBackColor

Description

The "ScaleBackColor" property specifies the background color of the scale.

Type

UInt32

Access

Read-write

Syntax

```
Slider.ScaleBackColor
```

See also

Slider (Page 5563)

Slider.ScaleForeColor (Page 5599)

Slider.ScaleForeColor

Description

The "ScaleForeColor" property specifies the foreground color of the scale.

Type

UInt32

Access

Read-write

Syntax

```
Slider.ScaleForeColor
```

See also

Slider (Page 5563)

Slider.ScaleBackColor (Page 5599)

Slider.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the slider is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Slider.ShowFocusVisual
```

See also

Slider (Page 5563)

Slider.ShowTrendIndicator

Description

The "ShowTrendIndicator" property specifies whether the tendency (rising or falling) of the process value to be monitored is indicated by means of a small arrow.

Type

Bool

Access

Read-write

Syntax

```
Slider.ShowTrendIndicator
```

See also

Slider (Page 5563)

Slider.ShowValue

Description

The "ShowValue" property specifies whether the process value is additionally output as text.

Type

Bool

Access

Read-write

Syntax

```
Slider.ShowValue
```

See also

Slider (Page 5563)

Slider.StraightScale

Description

The "StraightScale" property specifies the scale of the slider.

Type

Object, HmiStraightScalePart

Access

Read-write

Syntax

```
Slider.StraightScale
```

See also

Slider (Page 5563)

StraightScale.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
StraightScale.AutoScaling
```

See also

Slider.StraightScale (Page 5601)

StraightScale.BeginValue

Description

The "BeginValue" property specifies the start of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
StraightScale.BeginValue
```

See also

Slider.StraightScale (Page 5601)

StraightScale.DivisionCount

Description

The "DivisionCount" property specifies the number of main units with subdivisions. To this purpose the automatic scaling must be switched off.

Type

Int32

Access

Read-write

Syntax

```
StraightScale.DivisionCount
```

See also

Slider.StraightScale (Page 5601)

StraightScale.EndValue

Description

The "EndValue" property specifies the end of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
StraightScale.EndValue
```

See also

Slider.StraightScale (Page 5601)

StraightScale.LabelColor

Description

The "LabelColor" property specifies the color of the labeling.

Type

UInt32

Access

Read-write

Syntax

```
StraightScale.LabelColor
```

See also

Slider.StraightScale (Page 5601)

StraightScale.LabelFont

Description

The "LabelFont" property specifies the font of the labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StraightScale.LabelFont
```

See also

Slider.StraightScale (Page 5601)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StraightScale.LabelFont (Page 5604)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StraightScale.LabelFont (Page 5604)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StraightScale.LabelFont (Page 5604)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StraightScale.LabelFont (Page 5604)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StraightScale.LabelFont (Page 5604)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StraightScale.LabelFont (Page 5604)

StraightScale.LargeTickLabelingStep

Description

The "LargeTickLabelingStep" property specifies the interval at which scale sections are labeled.

Type

UInt8

Access

Read-write

Syntax

```
StraightScale.LargeTickLabelingStep
```

See also

Slider.StraightScale (Page 5601)

StraightScale.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
StraightScale.MeasurementUnit
```

See also

Slider.StraightScale (Page 5601)

StraightScale.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
StraightScale.MeasurementUnitType
```

See also

Slider.StraightScale (Page 5601)

StraightScale.Orientation

Description

The "Orientation" property specifies the orientation of the scale.

Type

Int32, HmiOrientation

Specifies the alignment:

- Horizontal (0): Horizontal
- Vertical (1): Vertical

Access

Read-write

Syntax

```
StraightScale.Orientation
```

See also

Slider.StraightScale (Page 5601)

StraightScale.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the process values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
StraightScale.OutputFormat
```

See also

Slider.StraightScale (Page 5601)

StraightScale.ScaleMode

Description

The "ScaleMode" property specifies the type of scaling.

Type

Int32, HmiScaleMode

Specifies the scaling:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
StraightScale.ScaleMode
```

See also

Slider.StraightScale (Page 5601)

StraightScale.ScalingType

Description

The "ScalingType" property specifies the scaling.

Type

Int32, HmiScalingType

Specifies the scaling:

- Linear (0): Linear
- Logarithmic (1): Logarithmic
- NegativeLogarithmic (2): Negative logarithmic
- Tangent (4): Tangential
- Quadratic (5): Square
- Cubic (6): Cubic

Access

Read-write

Syntax

```
StraightScale.ScalingType
```

See also

Slider.StraightScale (Page 5601)

StraightScale.SubDivisionCount

Description

The "SubDivisionCount" property specifies the number of subdivisions of the main units.

Type

Int32

Access

Read-write

Syntax

```
StraightScale.SubDivisionCount
```

See also

Slider.StraightScale (Page 5601)

StraightScale.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
StraightScale.TickColor
```

See also

Slider.StraightScale (Page 5601)

Slider.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the slider.

Type

String

Access

Read-only

Syntax

```
Slider.StyleItemClass
```

See also

Slider (Page 5563)

Slider.TabIndex

Description

The "TabIndex" property returns the position of the slider in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Slider.TabIndex
```

See also

Slider (Page 5563)

Slider.ThresholdIndicators

Description

The "ThresholdIndicators" property specifies how parameterized limit values are visualized.

Type

Int32, HmiThresholdIndicator

Specifies the visualization:

- None (0): None
- Lines (1): Lines
- Markers (2): Markers

Access

Read-write

Syntax

```
Slider.ThresholdIndicators
```

See also

Slider (Page 5563)

Slider.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the slider.

Type

Object, HmiThresholdCollection (Page 5615)

Access

Read-only

Syntax

```
Slider.Thresholds
```

See also

Slider (Page 5563)

HmiThresholdCollection (Page 5615)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

Slider.Thresholds (Page 5614)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 5615)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 5617)

See also

HmiThresholdCollection (Page 5615)

Threshold (Page 5617)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.
- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

See also

HmiThresholdCollection (Page 5615)

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 5617)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 5617)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 5617)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold
- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 5617)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 5617)

Slider.ThumbBackColor

Description

The "ThumbBackColor" property specifies the background color of the handle of the slider.

Type

UInt32

Access

Read-write

Syntax

```
Slider.ThumbBackColor
```

See also

Slider (Page 5563)

Slider.ThumbForeColor (Page 5620)

Slider.ThumbForeColor

Description

The "ThumbForeColor" property specifies the foreground color of the handle of the slider.

Type

UInt32

Access

Read-write

Syntax

```
Slider.ThumbForeColor
```

See also

Slider (Page 5563)

Slider.ThumbBackColor (Page 5620)

Slider.Title

Description

The "Title" property specifies the caption that appears as the title.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
Slider.Title
```

See also

Slider (Page 5563)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Slider.Title (Page 5621)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5621)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5621)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5621)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5621)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5621)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5621)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Slider.Title (Page 5621)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Slider.Title (Page 5621)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Slider.Title (Page 5621)

Slider.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Slider.ToolTipText
```

See also

Slider (Page 5563)

Slider.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Slider.Top
```

See also

Slider (Page 5563)

Slider.TrendIndicatorColor

Description

The "TrendIndicatorColor" property specifies the color of the trend indicator. The trend indicator uses a small arrow to represent the tendency (rising or falling) of the process value to be monitored. To activate the trend indicator, the "ShowTrendIndicator" property must be activated.

Type

UInt32

Access

Read-write

Syntax

```
Slider.TrendIndicatorColor
```

See also

Slider (Page 5563)

Slider.ValuePosition

Description

The "ValuePosition" property specifies where the value of the current slider position is additionally displayed numerically.

Type

Int32, HmiSimplePosition

Specifies the position:

- LeftOrTop (0): Left for vertical alignment, top for horizontal alignment
- RightOrBottom (1): Right for vertical alignment, bottom for horizontal alignment

Access

Read-write

Syntax

```
Slider.ValuePosition
```

See also

Slider (Page 5563)

Slider.Visible

Description

The "Visible" property specifies whether the slider is visible.

Type

Bool

Access

Read-write

Syntax

```
Slider.Visible
```

See also

Slider (Page 5563)

Slider.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
Slider.VisualizeQuality
```

See also

Slider (Page 5563)

Slider.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Slider.Width
```

See also

Slider (Page 5563)

Slider.WriteDuringChange

Description

The "WriteDuringChange" property specifies when changes are transferred.

- True: Even during the modification of the slider.
- False: Only after the slider has been released.

Type

Bool

Access

Read-write

Syntax

```
Slider.WriteDuringChange
```

See also

Slider (Page 5563)

Slider.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the slider.

Syntax

```
Slider.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Slider (Page 5563)

Slider.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Slider.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Slider (Page 5563)

Slider_OnActivated()

Description

The "OnActivated" event occurs when a slider receives focus:

- A slider is selected via the configured tab sequence.
- A slider that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Slider_OnActivated(item)
```

Context

item

Type: Object

Slider where the event occurs.

See also

Slider (Page 5563)

Slider_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A slider is right-clicked.
- A slider is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Slider_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Slider where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Slider (Page 5563)

Slider_OnDeactivated()

Description

The "OnDeactivated" event occurs when the slider loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Slider_OnDeactivated(item)
```

Context

item

Type: Object

Slider where the event occurs.

See also

Slider (Page 5563)

Slider_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the slider is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Slider_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Slider where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Slider (Page 5563)

Slider_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the slider is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Slider_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Slider where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Slider (Page 5563)

Slider_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A slider is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a slider has the focus.
- A slider is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Slider_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Slider where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Slider (Page 5563)

SwacContainer

Description

The "SwacContainer" object represents a container for SWAC components (Siemens Web Application Collaboration) in runtime.

Object type

HmiSwacContainer

Properties

The "SwacContainer" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the container.
- **Enabled**
  Specifies whether the container can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the container.
- **Layer**
  Returns the screen layer in which the container is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the container.
- **Operability**
  Returns whether the container is operable.
- **Parent**
  Returns the higher-level screen object.
- **Properties**
  allows access to the dynamic properties of the SWAC component.
- **RenderingTemplate**
  Returns the name of the template from which the container was created.
- **RequireExplicitUnlock**
  Returns whether the container is only operable while the corresponding button is being pressed.
- **ShowFocusVisual**
  Specifies whether the container is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the container.

- **TabIndex**
  Returns the position of the container in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **Url**
  Specifies the URL displayed by the container.
- **Visible**
  Specifies whether the container is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the container.

Methods

The "SwacContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the container.
- **PropertyFlashing()**
  Configures flashing of a property.

SwacContainer.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
SwacContainer.Authorization
```

See also

SwacContainer (Page 5639)

SwacContainer.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
SwacContainer.Caption
```

See also

SwacContainer (Page 5639)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

SwacContainer.Caption (Page 5642)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5642)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5642)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5642)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5642)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5642)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5642)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

SwacContainer.Caption (Page 5642)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

SwacContainer.Caption (Page 5642)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

SwacContainer.Caption (Page 5642)

SwacContainer.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
SwacContainer.CaptionColor
```

See also

SwacContainer (Page 5639)

SwacContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects (CustomControl, SwacComponent, or WidgetType).

Type

String

Access

Read-only

Syntax

```
SwacContainer.ContainedType
```

See also

SwacContainer (Page 5639)

SwacContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the container.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
SwacContainer.CurrentQuality
```

See also

SwacContainer (Page 5639)

SwacContainer.Enabled

Description

The "Enabled" property specifies whether the container can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
SwacContainer.Enabled
```

See also

SwacContainer (Page 5639)

SwacContainer.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SwacContainer.Height
```

See also

SwacContainer (Page 5639)

SwacContainer.Icon

Description

The "Icon" property specifies the icon of the container.

Type

String

Access

Read-write

Syntax

```
SwacContainer.Icon
```

See also

SwacContainer (Page 5639)

SwacContainer.Layer

Description

The "Layer" property returns the screen layer in which the container is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
SwacContainer.Layer
```

See also

SwacContainer (Page 5639)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

SwacContainer.Layer (Page 5651)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

SwacContainer.Layer (Page 5651)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

SwacContainer.Layer (Page 5651)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

SwacContainer.Layer (Page 5651)

SwacContainer.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
SwacContainer.Left
```

See also

SwacContainer (Page 5639)

SwacContainer.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
SwacContainer.Margin
```

See also

SwacContainer (Page 5639)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

SwacContainer.Margin (Page 5654)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

SwacContainer.Margin (Page 5654)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

SwacContainer.Margin (Page 5654)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

SwacContainer.Margin (Page 5654)

SwacContainer.Name

Description

The "Name" property returns the name of the container.

Type

String

Access

Read-only

Syntax

```
SwacContainer.Name
```

See also

SwacContainer (Page 5639)

SwacContainer.Operability

Description

The "Operability" property returns whether the container is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
SwacContainer.Operability
```

See also

SwacContainer (Page 5639)

SwacContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
SwacContainer.Parent
```

See also

SwacContainer (Page 5639)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

SwacContainer.Properties

Description

The "Properties" property allows access to the dynamic properties of the SWAC component.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
SwacContainer.Properties
```

See also

SwacContainer (Page 5639)

SwacContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the container was created.

Type

String

Access

Read-only

Syntax

```
SwacContainer.RenderingTemplate
```

See also

SwacContainer (Page 5639)

SwacContainer.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the container is only operable while the corresponding button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
SwacContainer.RequireExplicitUnlock
```

See also

SwacContainer (Page 5639)

SwacContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the container is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
SwacContainer.ShowFocusVisual
```

See also

SwacContainer (Page 5639)

SwacContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the container.

Type

String

Access

Read-only

Syntax

```
SwacContainer.StyleItemClass
```

See also

SwacContainer (Page 5639)

SwacContainer.TabIndex

Description

The "TabIndex" property returns the position of the container in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
SwacContainer.TabIndex
```

See also

SwacContainer (Page 5639)

SwacContainer.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
SwacContainer.Top
```

See also

SwacContainer (Page 5639)

SwacContainer.Url

Description

The "Url" property specifies the URL displayed by the container.

Type

String

Access

Read-write

Syntax

```
SwacContainer.Url
```

See also

SwacContainer (Page 5639)

SwacContainer.Visible

Description

The "Visible" property specifies whether the container is visible.

Type

Bool

Access

Read-write

Syntax

```
SwacContainer.Visible
```

See also

SwacContainer (Page 5639)

SwacContainer.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SwacContainer.Width
```

See also

SwacContainer (Page 5639)

SwacContainer.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the container.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
SwacContainer.WindowFlags
```

Example

Adapting the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

SwacContainer (Page 5639)

SwacContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the container.

Syntax

```
SwacContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

SwacContainer (Page 5639)

SwacContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
SwacContainer.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

SwacContainer (Page 5639)

SymbolContainer

Description

The "SymbolContainer" object represents a container for displaying the project graphics in runtime.

Object type

HmiSymbolContainer

Properties

The "SymbolContainer" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **ContainedType**
  Returns the type of the contained objects.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the container.
- **Effect**
  Specifies the effect for the display of the container surface.
- **EffectColor**
  Specifies the color of the container surface.
- **Enabled**
  Specifies whether the container can be operated in runtime.
- **Flip**
  Specifies whether the symbol is mirrored.
- **Height**
  Specifies the height.
- **Layer**
  Returns the screen layer in which the container is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the container.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the container is operable.

- **Parent**
  Returns the higher-level screen object.
- **ProcessValue**
  Specifies the process value.
- **Properties**
  Allows access to the dynamic properties of the project graphics.
- **RenderingTemplate**
  Returns the name of the template from which the container was created.
- **RequireExplicitUnlock**
  Returns whether the container is only operable while the corresponding button is being pressed.
- **RotationAngle**
  Specifies the rotation angle in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the container rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the container is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the container.
- **TabIndex**
  Returns the position of the container in the tab sequence.
- **Thresholds**
  Returns the list of all limit values of the container.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the container is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "SymbolContainer" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the container.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "SymbolContainer" object has the following events:

- **OnContextTapped()**
  Occurs when a container is right-clicked or long-touched.
- **OnKeyDown()**
  Occurs when a key is pressed while the container is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the container is in focus.
- **OnTapped()**
  Occurs when a container is left-clicked or short-touched.

SymbolContainer.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
SymbolContainer.Authorization
```

See also

SymbolContainer (Page 5666)

SymbolContainer.ContainedType

Description

The "ContainedType" property returns the type of the contained objects (CustomControl, SwacComponent, or WidgetType).

Type

String

Access

Read-only

Syntax

```
SymbolContainer.ContainedType
```

See also

SymbolContainer (Page 5666)

SymbolContainer.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the container.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
SymbolContainer.CurrentQuality
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Effect

Description

The "Effect" property specifies the effect for the display of the container surface.

Type

Int32, HmiSymbolEffect

Specifies the effect.

- None (0): None
- UseEffectColor (1): Uses the "EffectColor".

Access

Read-write

Syntax

```
SymbolContainer.Effect
```

See also

SymbolContainer (Page 5666)

SymbolContainer.EffectColor

Description

The "EffectColor" property specifies the color of the container surface.

Type

UInt32

Access

Read-write

Syntax

```
SymbolContainer.EffectColor
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Enabled

Description

The "Enabled" property specifies whether the container can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
SymbolContainer.Enabled
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Flip

Description

The "Flip" property specifies whether the symbol is mirrored.

Type

Int32, HmiFlipMode

Specifies the mirroring:

- None (0): None
- Horizontal (1): Horizontal (mirrored at vertical axis)

Access

Read-write

Syntax

```
SymbolContainer.Flip
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SymbolContainer.Height
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Layer

Description

The "Layer" property returns the screen layer in which the container is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
SymbolContainer.Layer
```

See also

SymbolContainer (Page 5666)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

SymbolContainer.Layer (Page 5672)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

SymbolContainer.Layer (Page 5672)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

SymbolContainer.Layer (Page 5672)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

SymbolContainer.Layer (Page 5672)

SymbolContainer.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
SymbolContainer.Left
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
SymbolContainer.Margin
```

See also

SymbolContainer (Page 5666)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

SymbolContainer.Margin (Page 5675)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

SymbolContainer.Margin (Page 5675)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

SymbolContainer.Margin (Page 5675)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

SymbolContainer.Margin (Page 5675)

SymbolContainer.Name

Description

The "Name" property returns the name of the container.

Type

String

Access

Read-only

Syntax

```
SymbolContainer.Name
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Opacity

Description

The "Opacity" property specifies the opacity. The "0" value indicates completely transparency.

Type

Float

Access

Read-write

Syntax

```
SymbolContainer.Opacity
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Operability

Description

The "Operability" property returns whether the container is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
SymbolContainer.Operability
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
SymbolContainer.Parent
```

See also

SymbolContainer (Page 5666)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

SymbolContainer.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
SymbolContainer.ProcessValue
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Properties

Description

The "Properties" property allows access to the dynamic properties of the project graphics.

Type

Object, HmiDynamicPropertyPart

Access

Read-write

Syntax

```
SymbolContainer.Properties
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the container was created.

Type

String

Access

Read-only

Syntax

```
SymbolContainer.RenderingTemplate
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the container is only operable while the corresponding button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
SymbolContainer.RequireExplicitUnlock
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle in degrees.

Type

Int16

Access

Read-write

Syntax

```
SymbolContainer.RotationAngle
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RotationCenterPlacement (Page 5682)

SymbolContainer.RotationCenterX (Page 5683)

SymbolContainer.RotationCenterY (Page 5684)

SymbolContainer.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the container rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in the DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
SymbolContainer.RotationCenterPlacement
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RotationAngle (Page 5682)

SymbolContainer.RotationCenterX (Page 5683)

SymbolContainer.RotationCenterY (Page 5684)

SymbolContainer.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
SymbolContainer.RotationCenterX
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RotationAngle (Page 5682)

SymbolContainer.RotationCenterPlacement (Page 5682)

SymbolContainer.RotationCenterY (Page 5684)

SymbolContainer.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
SymbolContainer.RotationCenterY
```

See also

SymbolContainer (Page 5666)

SymbolContainer.RotationAngle (Page 5682)

SymbolContainer.RotationCenterPlacement (Page 5682)

SymbolContainer.RotationCenterX (Page 5683)

SymbolContainer.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the container is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
SymbolContainer.ShowFocusVisual
```

See also

SymbolContainer (Page 5666)

SymbolContainer.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the container.

Type

String

Access

Read-only

Syntax

```
SymbolContainer.StyleItemClass
```

See also

SymbolContainer (Page 5666)

SymbolContainer.TabIndex

Description

The "TabIndex" property returns the position of the container in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
SymbolContainer.TabIndex
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the container.

Type

Object, HmiThresholdCollection (Page 5686)

Access

Read-only

Syntax

```
SymbolContainer.Thresholds
```

See also

SymbolContainer (Page 5666)

HmiThresholdCollection (Page 5686)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

SymbolContainer.Thresholds (Page 5686)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 5686)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 5688)

See also

HmiThresholdCollection (Page 5686)

Threshold (Page 5688)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.

- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

See also

HmiThresholdCollection (Page 5686)

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 5688)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 5688)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 5688)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold

- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 5688)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 5688)

SymbolContainer.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
SymbolContainer.ToolTipText
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
SymbolContainer.Top
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Visible

Description

The "Visible" property specifies whether the container is visible.

Type

Bool

Access

Read-write

Syntax

```
SymbolContainer.Visible
```

See also

SymbolContainer (Page 5666)

SymbolContainer.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
SymbolContainer.VisualizeQuality
```

See also

SymbolContainer (Page 5666)

SymbolContainer.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SymbolContainer.Width
```

See also

SymbolContainer (Page 5666)

SymbolContainer.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the container.

Syntax

```
SymbolContainer.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

SymbolContainer (Page 5666)

SymbolContainer.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
SymbolContainer.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

SymbolContainer (Page 5666)

SymbolContainer_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A container is right-clicked.
- A container is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
SymbolContainer_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Container where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key

- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

SymbolContainer (Page 5666)

SymbolContainer_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the container is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
SymbolContainer_OnKeyDown(item, keyCode, modifiers)
```

Context

`item`

Type: Object

Container where the event occurs.

`keyCode`

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

SymbolContainer (Page 5666)

SymbolContainer_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the container is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
SymbolContainer_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Container where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

SymbolContainer (Page 5666)

SymbolContainer_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A container type is left-clicked.
- The <RETURN> or <SPACE> key is pressed when an container has the focus.
- A container is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
SymbolContainer_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Container where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

SymbolContainer (Page 5666)

SymbolicIOField

Description

The "SymbolicIOField" object represents a symbolic I/O field for the input and display of text or graphic objects as a list in runtime.

Object type

HmiSymbolicIOField

Properties

The "SymbolicIOField" object has the following properties:

- **AcceptExplicitely**
  Specifies whether the process value is only written by explicit triggering of double-click, enter key, or tab.
- **AcceptOnDeactivated**
  Specifies whether the process value is written when the symbolic I/O field loses the input focus.
- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.

- **AutoTabOnAccept**
  Specifies whether a switch to the next object in the configured tab sequence occurs automatically after a value is entered.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **Content**
  Specifies the display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the symbolic I/O field.
- **Enabled**
  Specifies whether the symbolic I/O field can be operated in runtime.
- **ExpandOnActivate**
  Specifies whether the list drops down when the symbolic I/O field receives focus.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Graphic**
  Returns the graphic of the graphics list according to the process value.
- **Height**
  Specifies the height.
- **IOFieldType**
  Specifies the mode of the symbolic I/O field.
- **Layer**
  Returns the layer of the screen in which the symbolic I/O field is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the symbolic I/O field.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the symbolic I/O field is operable.
- **Padding**
  Specifies the distance of the content from the border of the symbolic I/O field.

- **Parent**
  Returns the higher-level screen object.
- **ProcessValue**
  Specifies the process value.
- **RenderingTemplate**
  Returns the name of the template from which the symbolic I/O field was created.
- **RequireExplicitUnlock**
  Returns whether the symbolic I/O field is only operable while the associated button is being pressed.
- **ResourceList**
  Specifies the text or graphics list.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the symbolic I/O field rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **SelectedIndex**
  Specifies the index for the selected list entry.
- **SelectionBackColor**
  Specifies the background color of the selected list entry.
- **SelectionForeColor**
  Specifies the font color of the text of the selected list entry.
- **ShowFocusVisual**
  Specifies whether the symbolic I/O field is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the symbolic I/O field.
- **TabIndex**
  Returns the position of the symbolic I/O field in the tab sequence.
- **Text**
  Returns the text of the text list according to the process value.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the symbolic I/O field is visible.

- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "SymbolicIOField" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the symbolic I/O field.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "SymbolicIOField" object has the following events:

- **OnActivated()**
  Occurs when a symbolic I/O field receives focus.
- **OnContextTapped()**
  Occurs when a symbolic I/O field is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a symbolic I/O field loses the focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the symbolic I/O field is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the symbolic I/O field is in focus.
- **OnTapped()**
  Occurs when a symbolic I/O field is left-clicked or short-touched.

SymbolicIOField.AcceptExplicitely

Description

The "AcceptExplicitely" property specifies whether the process value is only written by explicit triggering of double-click, Enter key, or tab.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.AcceptExplicitely
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.AcceptOnDeactivated

Description

The "AcceptOnDeactivated" property specifies whether the process value is written when the symbolic I/O field loses the input focus.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.AcceptOnDeactivated
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.AlternateBackColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.AlternateBorderColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
SymbolicIOField.Authorization
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.AutoTabOnAccept

Description

The "AutoTabOnAccept" property specifies whether a switch to the next object in the configured tab sequence occurs automatically after a value is entered.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.AutoTabOnAccept
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.BackColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.BorderColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
SymbolicIOField.BorderWidth
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
SymbolicIOField.Content
```

See also

SymbolicIOField (Page 5701)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

SymbolicIOField.Content (Page 5709)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

SymbolicIOField.Content (Page 5709)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

SymbolicIOField.Content (Page 5709)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

SymbolicIOField.Content (Page 5709)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

SymbolicIOField.Content (Page 5709)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

SymbolicIOField.Content (Page 5709)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

SymbolicIOField.Content (Page 5709)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

SymbolicIOField.Content (Page 5709)

SymbolicIOField.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the symbolic I/O field.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
SymbolicIOField.CurrentQuality
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Enabled

Description

The "Enabled" property specifies whether the symbolic I/O field can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.Enabled
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.ExpandOnActivate

Description

The "ExpandOnActivate" property specifies whether the list drops down when the symbolic I/O field receives focus.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.ExpandOnActivate
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
SymbolicIOField.Font
```

See also

SymbolicIOField (Page 5701)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

SymbolicIOField.Font (Page 5716)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

SymbolicIOField.Font (Page 5716)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

SymbolicIOField.Font (Page 5716)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

SymbolicIOField.Font (Page 5716)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

SymbolicIOField.Font (Page 5716)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

SymbolicIOField.Font (Page 5716)

SymbolicIOField.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.ForeColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Graphic

Description

The "Graphic" property returns the graphic of the graphics list according to the process value.

Type

String

Access

Read-only

Syntax

```
SymbolicIOField.Graphic
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.Height
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.IOFieldType

Description

The "IOFieldType" property specifies the mode of the symbolic I/O field.

Type

Int32, HmiIOFieldType

Specifies the type:

- Output (0): Output only
- InputOutput (2): Input and output

Access

Read-write

Syntax

```
SymbolicIOField.IOFieldType
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Layer

Description

The "Layer" property returns the layer of the screen in which the symbolic I/O field is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
SymbolicIOField.Layer
```

See also

SymbolicIOField (Page 5701)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

SymbolicIOField.Layer (Page 5721)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

SymbolicIOField.Layer (Page 5721)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

SymbolicIOField.Layer (Page 5721)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

SymbolicIOField.Layer (Page 5721)

SymbolicIOField.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
SymbolicIOField.Left
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
SymbolicIOField.Margin
```

See also

SymbolicIOField (Page 5701)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

SymbolicIOField.Margin (Page 5724)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

SymbolicIOField.Margin (Page 5724)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

SymbolicIOField.Margin (Page 5724)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

SymbolicIOField.Margin (Page 5724)

SymbolicIOField.Name

Description

The "Name" property returns the name of the symbolic I/O field.

Type

String

Access

Read-only

Syntax

```
SymbolicIOField.Name
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
SymbolicIOField.Opacity
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Operability

Description

The "Operability" property returns whether the symbolic I/O field is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
SymbolicIOField.Operability
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Padding

Description

The "Padding" property specifies the distance of the content from the border of the symbolic I/O field.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
SymbolicIOField.Padding
```

See also

SymbolicIOField (Page 5701)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

SymbolicIOField.Padding (Page 5728)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

SymbolicIOField.Padding (Page 5728)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

SymbolicIOField.Padding (Page 5728)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

SymbolicIOField.Padding (Page 5728)

SymbolicIOField.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
SymbolicIOField.Parent
```

See also

SymbolicIOField (Page 5701)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

SymbolicIOField.ProcessValue

Description

The "ProcessValue" property specifies the process value.

Type

Variant

Access

Read-write

Syntax

```
SymbolicIOField.ProcessValue
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the symbolic I/O field was created.

Type

String

Access

Read-only

Syntax

```
SymbolicIOField.RenderingTemplate
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the symbolic I/O field can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
SymbolicIOField.RequireExplicitUnlock
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.ResourceList

Description

The "ResourceList" property specifies the text or graphics list.

Type

String, HmiResourceList

Access

Read-write

Syntax

```
SymbolicIOField.ResourceList
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
SymbolicIOField.RotationAngle
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RotationCenterPlacement (Page 5734)

SymbolicIOField.RotationCenterX (Page 5735)

SymbolicIOField.RotationCenterY (Page 5736)

SymbolicIOField.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the symbolic I/O field rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
SymbolicIOField.RotationCenterPlacement
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RotationAngle (Page 5734)

SymbolicIOField.RotationCenterX (Page 5735)

SymbolicIOField.RotationCenterY (Page 5736)

SymbolicIOField.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
SymbolicIOField.RotationCenterX
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RotationAngle (Page 5734)

SymbolicIOField.RotationCenterPlacement (Page 5734)

SymbolicIOField.RotationCenterY (Page 5736)

SymbolicIOField.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
SymbolicIOField.RotationCenterY
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.RotationAngle (Page 5734)

SymbolicIOField.RotationCenterPlacement (Page 5734)

SymbolicIOField.RotationCenterX (Page 5735)

SymbolicIOField.SelectedIndex

Description

The "SelectedIndex" property specifies the index of the selected list entry. Returns the index according to the process value. The index is "-1", if no selection is available or the process value does not match.

Type

Int32

Access

Read-write

Syntax

```
SymbolicIOField.SelectedIndex
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected list entry

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.SelectionBackColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the text of the selected list entry.

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.SelectionForeColor
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the symbolic I/O field is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.ShowFocusVisual
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the symbolic I/O field.

Type

String

Access

Read-only

Syntax

```
SymbolicIOField.StyleItemClass
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.TabIndex

Description

The "TabIndex" property returns the position of the symbolic I/O field in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
SymbolicIOField.TabIndex
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Text

Description

The "Text" property returns the text of the text list according to the process value.

Type

String

Access

Read-only

Syntax

```
SymbolicIOField.Text
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
SymbolicIOField.ToolTipText
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
SymbolicIOField.Top
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Visible

Description

The "Visible" property specifies whether the symbolic I/O field is visible.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.Visible
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
SymbolicIOField.VisualizeQuality
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SymbolicIOField.Width
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the symbolic I/O field.

Syntax

```
SymbolicIOField.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

SymbolicIOField (Page 5701)

SymbolicIOField.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
SymbolicIOField.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

SymbolicIOField (Page 5701)

SymbolicIOField_OnActivated()

Description

The "OnActivated" event occurs when a symbolic I/O field receives focus:

- A symbolic I/O field is selected via the configured tab sequence.
- A symbolic I/O field that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
SymbolicIOField_OnActivated(item)
```

Context

item

Type: Object

Symbolic I/O field where the event occurs.

See also

SymbolicIOField (Page 5701)

SymbolicIOField_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A symbolic I/O field is right-clicked.
- A symbolic I/O field is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
SymbolicIOField_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Symbolic I/O field where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key

- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

SymbolicIOField (Page 5701)

SymbolicIOField_OnDeactivated()

Description

The "OnDeactivated" event occurs when the symbolic I/O field loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
SymbolicIOField_OnDeactivated(item)
```

Context

`item`

Type: Object

Symbolic I/O field where the event occurs.

See also

SymbolicIOField (Page 5701)

SymbolicIOField_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the symbolic I/O field is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
SymbolicIOField_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Symbolic I/O field where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

SymbolicIOField (Page 5701)

SymbolicIOField_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the symbolic I/O field is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
SymbolicIOField_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Symbolic I/O field where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key

- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

SymbolicIOField (Page 5701)

SymbolicIOField_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A symbolic I/O field is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a symbolic I/O field has the focus.
- A symbolic I/O field is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
SymbolicIOField_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Symbolic I/O field where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

SymbolicIOField (Page 5701)

SystemDiagnosisControl

Description

The "SystemDiagnosisControl" object represents a system diagnostics control in runtime.

Object type

HmiSystemDiagnosisControl

Properties

The "SystemDiagnosisControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the system diagnostics control.
- **DiagnosisBufferPathText**
  Specifies the path to the diagnostic buffer.
- **DiagnosisOverviewPathText**
  Specifies the path to the diagnostics overview.
- **Enabled**
  Specifies whether the system diagnostics control can be operated in runtime.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the system diagnostics control.
- **Layer**
  Returns the screen layer in which the system diagnostics control is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **MatrixView**
  Specifies the matrix view of the system diagnostics control.
- **Name**
  Returns the name of the system diagnostics control.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the system diagnostics control was created.
- **ShowFocusVisual**
  Specifies whether the system diagnostics control is highlighted when in focus.
- **ShowStatusPath**
  Specifies that the status path is displayed.

- **StatusBar**
  Specifies the information bar of the system diagnostics control.
- **StyleItemClass**
  Returns the style which is applied to the system diagnostics control.
- **SystemDiagnosisView**
  Specifies the table layout.
- **SystemDiagnosisViewType**
  Specifies the type of table layout.
- **TabIndex**
  Returns the position of the system diagnostics control in the tab sequence.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Specifies the toolbar of the system diagnostics control.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the system diagnostics control is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the system diagnostics control.

Methods

The "SystemDiagnosisControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the system diagnostics control.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **GoToPlc()**
  Navigates to the next PLC.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "SystemDiagnosisControl" object has the following events:

- **OnActivated()**
  Occurs when a system diagnostics control receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the system diagnostics control.

- **OnDeactivated()**
  Occurs when a system diagnostics control loses focus.
- **OnInitialized()**
  Occurs when a system diagnostics control has been successfully initialized and the data connection to the PLC has been established.

SystemDiagnosisControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControl.BackColor
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
SystemDiagnosisControl.Caption
```

See also

SystemDiagnosisControl (Page 5750)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

SystemDiagnosisControl.Caption (Page 5753)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 5754)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 5754)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 5754)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 5754)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 5754)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 5754)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

SystemDiagnosisControl.Caption (Page 5753)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

SystemDiagnosisControl.Caption (Page 5753)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

SystemDiagnosisControl.Caption (Page 5753)

SystemDiagnosisControl.CaptionColor

Description

The "CaptionColor" property specifies the background color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControl.CaptionColor
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the system diagnostics control.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
SystemDiagnosisControl.CurrentQuality
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.DiagnosisBufferPathText

Description

The "DiagnosisBufferPathText" property specifies the path to the diagnostic buffer.

Type

String

Access

Read-write

Syntax

```
SystemDiagnosisControl.DiagnosisBufferPathText
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.DiagnosisOverviewPathText

Description

The "DiagnosisOverviewPathText" property specifies the path to the diagnostics overview.

Type

String

Access

Read-write

Syntax

```
SystemDiagnosisControl.DiagnosisOverviewPathText
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Enabled

Description

The "Enabled" property specifies whether the system diagnostics control can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControl.Enabled
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Height

Description

The "Height" property specifies the height.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControl.Height
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Icon

Description

The "Icon" property specifies the icon of the system diagnostics control.

Type

String

Access

Read-write

Syntax

```
SystemDiagnosisControl.Icon
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Layer

Description

The "Layer" property returns the screen layer in which the system diagnostics control is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
SystemDiagnosisControl.Layer
```

See also

SystemDiagnosisControl (Page 5750)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

SystemDiagnosisControl.Layer (Page 5763)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

SystemDiagnosisControl.Layer (Page 5763)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

SystemDiagnosisControl.Layer (Page 5763)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

SystemDiagnosisControl.Layer (Page 5763)

SystemDiagnosisControl.Left

Description

The "Left" property specifies the value of the X coordinate.

Type

Int32

Access

Read-write

Syntax

```
SystemDiagnosisControl.Left
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
SystemDiagnosisControl.Margin
```

See also

SystemDiagnosisControl (Page 5750)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

SystemDiagnosisControl.Margin (Page 5766)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

SystemDiagnosisControl.Margin (Page 5766)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

SystemDiagnosisControl.Margin (Page 5766)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

SystemDiagnosisControl.Margin (Page 5766)

SystemDiagnosisControl.MatrixView

Description

The "MatrixView" property specifies the matrix view of the system diagnostics control.

Type

Object, HmiMatrixViewPart

Access

Read-write

Syntax

```
SystemDiagnosisControl.MatrixView
```

See also

SystemDiagnosisControl (Page 5750)

MatrixView.TileBorderWidth

Description

The "TileBorderWidth" property specifies the border width of the tile.

Type

UInt8

Access

Read-write

Syntax

```
MatrixView.TileBorderWidth
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)

MatrixView.TileHeightMax

Description

The "TileHeightMax" property specifies the maximum height of the tile.

Type

UInt16

Access

Read-write

Syntax

```
MatrixView.TileHeightMax
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)

MatrixView.TileHeightMin

Description

The "TileHeightMin" property specifies the minimum height of the tile.

Type

UInt16

Access

Read-write

Syntax

```
MatrixView.TileHeightMin
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)

MatrixView.TileWidthMax

Description

The "TileWidthMax" property specifies the maximum width of the tile.

Type

UInt16

Access

Read-write

Syntax

```
MatrixView.TileWidthMax
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)

MatrixView.TileWidthMin

Description

The "TileWidthMin" property specifies the minimum width of the tile.

Type

UInt16

Access

Read-write

Syntax

```
MatrixView.TileWidthMin
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)

MatrixView.HardwareDetails

Description

The "HardwareDetails" property returns a list of the HMI device hardware details.

Type

Object, HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

Access

Read-only

Syntax

```
MatrixView.HardwareDetails
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)

HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

HmiSystemDiagnosisHardwareDetailCollection

Description

The "HmiSystemDiagnosisHardwareDetailCollection" object is a list of all hardware details ("HmiSystemDiagnosisHardwareDetailPart" objects) of the HMI device on which the system diagnostics were activated.

You can reference the "HmiSystemDiagnosisHardwareDetailCollection" object via the "`MatrixView.HardwareDetails`" property.

Use

The "HmiSystemDiagnosisHardwareDetailCollection" object is a list which can be counted and enumerated. You can access the "HmiSystemDiagnosisHardwareDetailCollection" list using the index or the tag names.

Object type

HmiSystemDiagnosisHardwareDetailCollection

Properties

The "HmiSystemDiagnosisHardwareDetailCollection" object has the following properties:

- **Count**
  Returns the number of hardware details of the "HmiSystemDiagnosisHardwareDetailCollection" list.

Methods

The "HmiSystemDiagnosisHardwareDetailCollection" object has the following methods:

- **Item()**
  Returns the name of a hardware detail.

See also

MatrixView.HardwareDetails (Page 5771)

SystemDiagnosisControl.MatrixView (Page 5768)

HmiSystemDiagnosisHardwareDetailCollection.Count

Description

The "Count" property returns the number of hardware details of the "HmiSystemDiagnosisHardwareDetailCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiSystemDiagnosisHardwareDetailCollection.Count
```

See also

HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

MatrixView.HardwareDetails (Page 5771)

HmiSystemDiagnosisHardwareDetailCollection.Item()

Description

The "Item" method returns a hardware detail of the "HmiSystemDiagnosisHardwareDetailCollection" list.

Syntax

```
HmiSystemDiagnosisHardwareDetailCollection[.Item]
(HmiSystemDiagnosisHardwareDetailName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiSystemDiagnosisHardwareDetailCollection" object.

Parameters

HmiSystemDiagnosisHardwareDetailName

Type: String

Name of a hardware detail

Return value

Object, HmiSystemDiagnosisHardwareDetailPart (Page 5774)

See also

HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

SystemDiagnosisHardwareDetail (Page 5774)

SystemDiagnosisHardwareDetail

Description

The "SystemDiagnosisHardwareDetail" object represents a hardware detail.

Object type

HmiSystemDiagnosisHardwareDetailPart

Properties

The "SystemDiagnosisHardwareDetail" object has the following properties:

- **SystemDiagnosisMatrixBlock**
  Sets the matrix block of the system diagnostics control.
- **Visible**
  Specifies whether the hardware detail is visible.

Methods

--

See also

HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

MatrixView.HardwareDetails (Page 5771)

SystemDiagnosisHardwareDetail.SystemDiagnosisMatrixBlock

Description

The "SystemDiagnosisMatrixBlock" property specifies the matrix block of a hardware detail.

Type

Int32, HmiSystemDiagnosisMatrixBlock

Specifies the type of matrix block.

- Undefined (0): None
- Status (1): Status

- Name (2): Name
- OperatingState (3): Operating state
- Rack (4): Rack
- Slot (5): Slot
- OrderNumber (6): Order number
- Address (7): Address
- PlantDesignation (8): Plant Designation
- LocationIdentifier (9): Location identifier
- Subsystem (10): Subsystem
- Station (11): Station
- Subslot (12): Subslot
- SubAddress (13): Subaddress
- SoftwareVersion (14): Software Version
- Installation (15): Installation
- AdditionalInformation (16): Additional information
- ErrorDescription (17): Error description
- ManufacturerID (18): Manufacturer ID
- HardwareVersion (19): Hardware version
- ProfileID (20): Profile ID
- SpecificProfileData (21): Specific profile data
- IandMDataVersion (22): I and M Data Version
- SerialNumber (23): Serial number
- RevisionCounter (24): Version counter
- Type (25): Type
- IPAddress (32): IP addresses

Access

Read-write

Syntax

```
SystemDiagnosisHardwareDetail.SystemDiagnosisMatrixBlock
```

See also

SystemDiagnosisHardwareDetail (Page 5774)

HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

SystemDiagnosisHardwareDetail.Visible

Description

The "Visible" property specifies whether the hardware detail is visible.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisHardwareDetail.Visible
```

See also

SystemDiagnosisHardwareDetail (Page 5774)
HmiSystemDiagnosisHardwareDetailCollection (Page 5772)

MatrixView.SystemDiagnosisHardwareDetailView

Description

The "SystemDiagnosisHardwareDetailView" property specifies the view of the hardware detail.

Type

Object, HmiSystemDiagnosisDetailViewPart

Access

Read-write

Syntax

```
MatrixView.SystemDiagnosisHardwareDetailView
```

See also

SystemDiagnosisControl.MatrixView (Page 5768)
SystemDiagnosisControl (Page 5750)

SystemDiagnosisDetailView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.AllowFilter
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "True".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.AllowSort
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.AlternateBackColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the second foreground color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.AlternateForeColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.BackColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.CellPadding
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

SystemDiagnosisDetailView.CellPadding (Page 5779)
MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

SystemDiagnosisDetailView.CellPadding (Page 5779)
MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

SystemDiagnosisDetailView.CellPadding (Page 5779)
MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

SystemDiagnosisDetailView.CellPadding (Page 5779)

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.ColoringMode
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.Font
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

SystemDiagnosisDetailView.Font (Page 5782)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

SystemDiagnosisDetailView.Font (Page 5782)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

SystemDiagnosisDetailView.Font (Page 5782)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

SystemDiagnosisDetailView.Font (Page 5782)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

SystemDiagnosisDetailView.Font (Page 5782)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

SystemDiagnosisDetailView.Font (Page 5782)

SystemDiagnosisDetailView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.ForeColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.GridLineColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.GridLineVisibility

Description

The "GridLineVisibility" property specifies whether the grid lines are visible.

Type

Int32, HmiSimpleGridLine

Specifies the visibility of the grid lines:

- None (0): None
- Vertikal (1): Vertical grid lines visible
- Horizontal (2): Horizontal grid lines visible

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.GridLineVisibility
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.GridLineWidth
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.GridSelectionMode
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.HardwareDetails

Description

The "HardwareDetails" property returns a list of all hardware details.

Type

Object, HmiSystemDiagnosisHardwareDetailCollection (Page 5789)

Access

Read-only

Syntax

```
SystemDiagnosisDetailView.HardwareDetails
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

HmiSystemDiagnosisHardwareDetailCollection (Page 5789)

HmiSystemDiagnosisHardwareDetailCollection

Description

The "HmiSystemDiagnosisHardwareDetailCollection" object is a list of all hardware details ("HmiSystemDiagnosisHardwareDetailPart" objects) of the HMI device on which the system diagnostics were activated.

You can reference the "HmiSystemDiagnosisHardwareDetailCollection" object via the "`SystemDiagnosisDetailView.HardwareDetails`" property.

Use

The "HmiSystemDiagnosisHardwareDetailCollection" object is a list which can be counted and enumerated. You can access the "HmiSystemDiagnosisHardwareDetailCollection" list using the index or the tag names.

Object type

HmiSystemDiagnosisHardwareDetailCollection

Properties

The "HmiSystemDiagnosisHardwareDetailCollection" object has the following properties:

- **Count**
  Returns the number of hardware details of the "HmiSystemDiagnosisHardwareDetailCollection" list.

Methods

The "HmiSystemDiagnosisHardwareDetailCollection" object has the following methods:

- **Item()**
  Returns the name of a hardware detail.

See also

SystemDiagnosisDetailView.HardwareDetails (Page 5789)

HmiSystemDiagnosisHardwareDetailCollection.Count

Description

The "Count" property returns the number of hardware details of the "HmiSystemDiagnosisHardwareDetailCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiSystemDiagnosisHardwareDetailCollection.Count
```

See also

HmiSystemDiagnosisHardwareDetailCollection (Page 5789)

HmiSystemDiagnosisHardwareDetailCollection.Item()

Description

The "Item" method returns a hardware detail of the "HmiSystemDiagnosisHardwareDetailCollection" list.

Syntax

```
HmiSystemDiagnosisHardwareDetailCollection[.Item]
(HmiSystemDiagnosisHardwareDetailName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiSystemDiagnosisHardwareDetailCollection" object.

Parameters

HmiSystemDiagnosisHardwareDetailName

Type: String

Name of a hardware detail

Return value

Object, HmiSystemDiagnosisHardwareDetailPart (Page 5791)

See also

HmiSystemDiagnosisHardwareDetailCollection (Page 5789)

SystemDiagnosisHardwareDetail (Page 5791)

SystemDiagnosisHardwareDetail

Description

SystemDiagnosisHardwareDetail (Page 5774)

SystemDiagnosisDetailView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.HeaderSettings
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 5794)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 5794)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 5794)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 5794)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 5794)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 5794)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

SystemDiagnosisDetailView.HeaderSettings (Page 5791)

SystemDiagnosisDetailView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.HorizontalScrollBarVisibility
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.RowHeight
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.SelectFullRow
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.SelectionBackColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.SelectionBorderWidth

Description

The "SelectionBorderWidth" property specifies the border width of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.SelectionBorderWidth
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.SelectionForeColor
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisDetailView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
SystemDiagnosisDetailView.VerticalScrollBarVisibility
```

See also

MatrixView.SystemDiagnosisHardwareDetailView (Page 5776)

SystemDiagnosisControl.Name

Description

The "Name" property returns the name of the system diagnostics control.

Type

String

Access

Read-only

Syntax

```
SystemDiagnosisControl.Name
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 5750)

Access

Read-only

Syntax

```
SystemDiagnosisControl.Parent
```

See also

SystemDiagnosisControl (Page 5750)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

SystemDiagnosisControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the system diagnostics control was created.

Type

String

Access

Read-only

Syntax

```
SystemDiagnosisControl.RenderingTemplate
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the system diagnostics control is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControl.ShowFocusVisual
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.ShowStatusPath

Description

The "ShowStatusPath" property specifies that the status path is displayed.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControl.ShowStatusPath
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.StatusBar

Description

The "StatusBar" property specifies the information bar.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
SystemDiagnosisControl.StatusBar
```

See also

SystemDiagnosisControl (Page 5750)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 5809)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

HmiControlBarElementCollection (Page 5809)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 5809)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 5809)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 5811)

See also

HmiControlBarElementCollection (Page 5809)

Control Bar Elements (Page 5811)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5809)

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 5811)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 5811)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 5811)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 5811)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 5811)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 5817)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 5817)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 5817)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 5817)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 5817)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 5817)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 5817)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 5817)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 5811)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 5811)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 5811)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 5827)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 5827)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 5827)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 5827)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 5811)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 5811)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 5811)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 5811)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 5832)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 5832)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 5832)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 5832)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 5811)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 5811)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 5811)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5809)

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 5837)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 5839)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 5839)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 5839)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 5839)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 5839)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 5839)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 5839)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 5839)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment

- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms

- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 5837)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 5849)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 5849)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 5849)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 5849)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 5837)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 5854)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 5854)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 5854)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 5854)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 5837)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 5837)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5809)

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 5859)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 5866)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 5866)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 5866)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 5866)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 5859)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 5871)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 5871)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 5871)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 5871)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 5859)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 5859)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5809)

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line

- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time

- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 5877)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 5883)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 5883)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 5883)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 5883)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 5877)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 5888)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 5888)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 5888)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 5888)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 5877)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 5877)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.

- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5809)

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 5893)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 5902)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 5902)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 5902)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 5902)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 5893)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 5907)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 5907)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 5907)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 5907)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 5893)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 5893)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.

- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.

- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 5809)

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 5913)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 5920)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 5920)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 5913)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 5931)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 5931)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 5931)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 5931)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 5913)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 5936)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 5936)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 5936)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 5936)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 5913)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 5913)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 5942)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 5942)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 5942)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 5942)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 5942)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 5942)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 5945)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 5945)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 5945)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 5945)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 5948)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 5948)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 5948)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 5948)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

SystemDiagnosisControl.StatusBar (Page 5808)

SystemDiagnosisControl.StyleItemClass

Description

The "StyleItemClass" property returns the style that is applied to the system diagnostics control.

Type

String

Access

Read-only

Syntax

```
SystemDiagnosisControl.StyleItemClass
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.SystemDiagnosisView

Description

The "SystemDiagnosisView" property specifies the system diagnostics control.

Type

Object, HmiDataGridViewPart

Access

Read-write

Syntax

```
SystemDiagnosisControl.SystemDiagnosisView
```

See also

SystemDiagnosisControl (Page 5750)

DataGridView.AllowFilter

Description

The "AllowFilter" property specifies whether filtering of columns is permitted.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowFilter
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.AllowSort

Description

The "AllowSort" property specifies whether the sorting of columns is permitted.

- True: Activates the sorting and sets the "AllowSort" property of all columns to "true".
- False: Disables the sorting for all columns. The individual column properties remain unchanged.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.AllowSort
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateBackColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.AlternateForeColor

Description

The "AlternateForeColor" property specifies the flashing color for the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.AlternateForeColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.BackColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.CellPadding

Description

The "CellPadding" property specifies the inner spacing of the content from the cell border.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
DataGridView.CellPadding
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

DataGridView.CellPadding (Page 5955)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

DataGridView.CellPadding (Page 5955)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

DataGridView.CellPadding (Page 5955)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

DataGridView.CellPadding (Page 5955)

DataGridView.ColoringMode

Description

The "ColoringMode" property specifies whether alternate coloring of every other row or column is enabled.

Type

Int32, HmiGridColoringMode

Specifies the type of the coloring:

- None (0): None
- Columns (1): Color the columns alternately.
- Rows (2): Color the rows alternately.

Access

Read-write

Syntax

```
DataGridView.ColoringMode
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.Columns

Description

The "Columns" property represents the quantity of columns.

Type

Object, HmiDataGridColumnCollection (Page 5959)

Access

Read-only

Syntax

```
DataGridView.Columns
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)
HmiDataGridColumnCollection (Page 5959)

HmiDataGridColumnCollection

Description

The "HmiDataGridColumnCollection" object is a list of all columns ("DataGridColumn" objects) of the table.

You reference a "HmiDataGridColumnCollection" object via the `DataGridView.Columns` property.

Use

The "HmiDataGridColumnCollection" object is a list and can be counted and enumerated. You can access the "HmiDataGridColumnCollection" list using the index or the tag name.

Object type

HmiDataGridColumnCollection

Properties

The "HmiDataGridColumnCollection" object has the following properties:

- **Count**
  Returns the number of columns in the "HmiDataGridColumnCollection" list.

Methods

The "HmiDataGridColumnCollection" object has the following methods:

- **Item()**
  Returns a column of the "HmiDataGridColumnCollection" list.

See also

DataGridView.Columns (Page 5958)

HmiDataGridColumnCollection.Count

Description

The "Count" property returns the number of columns in the "HmiDataGridColumnCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiDataGridColumnCollection.Count
```

See also

HmiDataGridColumnCollection (Page 5959)

HmiDataGridColumnCollection.Item()

Description

The "Item" method returns a column of the "HmiDataGridColumnCollection" list.

Syntax

```
HmiDataGridColumnCollection[.Item](HmiDataGridColumnName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiDataGridColumnCollection" object.

Parameters

`HmiDataGridColumnName`

Type: String

Name of the column

Return value

Object, HmiDataGridColumnPartBase (Page 5961)

See also

HmiDataGridColumnCollection (Page 5959)

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn

Description

The "SystemDiagnosisControlColumn" object represents a value column.

Object type

HmiSystemDiagnosisControlColumnPart

Properties

The "SystemDiagnosisControlColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.
- **Content**
  Specifies the display options for text and graphics.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **SystemDiagnosisControlBlock**
  Specifies property that will be displayed in the column.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiDataGridColumnCollection (Page 5959)

HmiDataGridColumnCollection.Item() (Page 5960)

SystemDiagnosisControlColumn.AllowSort

Description

The "AllowSort" property specifies whether column sorting is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.AllowSort
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.BackColor
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.Content
```

See also

SystemDiagnosisControlColumn (Page 5961)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

SystemDiagnosisControlColumn.Content (Page 5963)

SystemDiagnosisControlColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.Enabled
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.ForeColor
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.Header
```

See also

SystemDiagnosisControlColumn (Page 5961)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

SystemDiagnosisControlColumn.Header (Page 5969)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 5970)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 5970)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

SystemDiagnosisControlColumn.Header (Page 5969)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

SystemDiagnosisControlColumn.Header (Page 5969)

SystemDiagnosisControlColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.MaximumWidth
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.MinimumWidth
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
SystemDiagnosisControlColumn.Name
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.OutputFormat
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.SortDirection

Description

The "SortDirection" property specifies the sorting direction.

Type

Int32, HmiSortDirection

Specifies the sorting order:

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.SortDirection
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.SortOrder

Description

The "SortOrder" property specifies the sorting order.

The sorting index begins at "`1`" (highest priority) in ascending order. `0` is ignored.

Type

UInt16

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.SortOrder
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.SystemDiagnosisControlBlock

Description

The "SystemDiagnosisControlBlock" specifies the property that will be displayed in the column.

Type

Int32, HmiSystemDiagnosisControlBlock

Specifies the control block:

- Undefined (0): Not defined
- Number (1): Number
- DateTime (2): Time information
- EventMessage (3): Event message
- EventType (4): Event type
- EventState (5): Event state

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.SystemDiagnosisControlBlock
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.Visible
```

See also

SystemDiagnosisControlColumn (Page 5961)

SystemDiagnosisControlColumn.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControlColumn.Width
```

See also

SystemDiagnosisControlColumn (Page 5961)

DataGridView.Font

Description

The "Font" property specifies the font of the cells.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridView.Font
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridView.Font (Page 5981)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridView.Font (Page 5981)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridView.Font (Page 5981)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridView.Font (Page 5981)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridView.Font (Page 5981)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridView.Font (Page 5981)

DataGridView.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.ForeColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.GridLineColor

Description

The "GridLineColor" property specifies the color of grid lines.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.GridLineColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.GridLineVisibility

Description

The "GridLineVisibility" property specifies the visibility of the grid lines.

Type

Int32, HmiSimpleGridLine

Specifies the grid lines:

- None (0): None
- Vertikal (1): Vertical
- Horizontal (2): Horizontal

Access

Read-write

Syntax

```
DataGridView.GridLineVisibility
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.GridLineWidth

Description

The "GridLineWidth" property specifies the thickness of the grid lines in pixels.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.GridLineWidth
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.GridSelectionMode

Description

The "GridSelectionMode" property specifies whether multiple selection is enabled in the table content.

Type

Int32, HmiGridSelectionMode

Specifies the type of the selection:

- None (0): None
- Single (1): Only single selection
- Multi (2): Multiple selection

Access

Read-write

Syntax

```
DataGridView.GridSelectionMode
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.HeaderSettings

Description

The "HeaderSettings" property specifies the settings for all column headers in the table.

Type

Object, HmiDataGridHeaderSettingsPart

Access

Read-write

Syntax

```
DataGridView.HeaderSettings
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridHeaderSettings.AllowColumnReorder

Description

The "AllowColumnReorder" property specifies whether the order of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnReorder
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.AllowColumnResize

Description

The "AllowColumnResize" property specifies whether the width of the columns can be changed.

Type

Bool

Access

Read-write

Syntax

```
DataGridHeaderSettings.AllowColumnResize
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.ColumnHeaderType

Description

The "ColumnHeaderType" property specifies the type of content of a column header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.ColumnHeaderType
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.Font

Description

The "Font" property specifies the font of the headers.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
DataGridHeaderSettings.Font
```

See also

DataGridView.HeaderSettings (Page 5987)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

DataGridHeaderSettings.Font (Page 5989)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

DataGridHeaderSettings.Font (Page 5989)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

DataGridHeaderSettings.Font (Page 5989)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

DataGridHeaderSettings.Font (Page 5989)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

DataGridHeaderSettings.Font (Page 5989)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

DataGridHeaderSettings.Font (Page 5989)

DataGridHeaderSettings.HeaderBackColor

Description

The "HeaderBackColor" property specifies the background color of the header.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderBackColor
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.HeaderForeColor

Description

The "HeaderForeColor" property specifies the font color of the headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderForeColor
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.HeaderGridLineColor

Description

The "HeaderGridLineColor" property specifies the color of the separation line between column headers.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderGridLineColor
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.HeaderSelectionBackColor

Description

The "HeaderSelectionBackColor" property specifies the background color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
DataGridHeaderSettings.HeaderSelectionBackColor
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.HeaderSelectionForeColor

Description

The "HeaderSelectionForeColor" property specifies the font color of the header of a selected line or column.

Type

UInt32

Access

Read-write

Syntax

```
Object.HeaderSelectionForeColor
```

`Object`

Required. An object from the "Availability" section.

See also

DataGridView.HeaderSettings (Page 5987)

DataGridHeaderSettings.RowHeaderType

Description

The "RowHeaderType" property specifies the type of content of a row header.

Type

Int32, HmiDataGridHeaderType

Specifies the type of headers:

- None (0): No column header
- Index (1): Consecutive number
- Content (2): Text and graphic

Access

Read-write

Syntax

```
DataGridHeaderSettings.RowHeaderType
```

See also

DataGridView.HeaderSettings (Page 5987)

DataGridView.HorizontalScrollBarVisibility

Description

The "HorizontalScrollBarVisibility" property specifies the visibility of the horizontal scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.HorizontalScrollBarVisibility
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.RowHeight

Description

The "RowHeight" property specifies the height of all rows in the table in DIU (Device Independent Unit).

"0" corresponds to a automatic mechanism, which adjusts the height of each line according to the font size and number of paragraphs.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.RowHeight
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.SelectFullRow

Description

The "SelectFullRow" property specifies whether only the cell or the entire row is included in the selection.

Type

Bool

Access

Read-write

Syntax

```
DataGridView.SelectFullRow
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.SelectionBackColor

Description

The "SelectionBackColor" property specifies the background color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionBackColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.SelectionBorderColor

Description

The "SelectionBorderColor" property specifies the border color of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.SelectionBorderWidth

Description

The "SelectionBorderWidth" property specifies the border width of the selected cells.

Type

UInt8

Access

Read-write

Syntax

```
DataGridView.SelectionBorderWidth
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.SelectionForeColor

Description

The "SelectionForeColor" property specifies the font color of the selected cells.

Type

UInt32

Access

Read-write

Syntax

```
DataGridView.SelectionForeColor
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

DataGridView.VerticalScrollBarVisibility

Description

The "VerticalScrollBarVisibility" property specifies the visibility of the vertical scroll bar of the display.

Type

Int32, HmiScrollBarVisibility

Specifies the visibility of the scroll bar:

- Automatic (0): Only visible if required
- Visible (1): Always visible
- Collapsed (2): Never visible

Access

Read-write

Syntax

```
DataGridView.VerticalScrollBarVisibility
```

See also

SystemDiagnosisControl.SystemDiagnosisView (Page 5952)

SystemDiagnosisControl.SystemDiagnosisViewType

Description

The "SystemDiagnosisViewType" property specifies the type of system diagnostics control.

Type

Int32, HmiSystemDiagnosisViewType

Specifies the type of system diagnostics control.

- Diagnosis (0): Diagnostic view
- Matrix (1): Matrix view

Access

Read-write

Syntax

```
SystemDiagnosisControl.SystemDiagnosisViewType
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.TabIndex

Description

The "TabIndex" property returns the position of the system diagnostics control in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
SystemDiagnosisControl.TabIndex
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.TimeZone

Description

The "TimeZone" property specifies the time zone.

Type

HmiTimeZone

Access

Read-write

Syntax

```
SystemDiagnosisControl.TimeZone
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.ToolBar

Description

The "ToolBar" property specifies the toolbar.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
SystemDiagnosisControl.ToolBar
```

See also

SystemDiagnosisControl (Page 5750)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 6004)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

HmiControlBarElementCollection (Page 6004)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 6003)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6004)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 5811)

See also

HmiControlBarElementCollection (Page 6004)

Control Bar Elements (Page 5811)

Control Bar Elements

Description

Control Bar Elements (Page 5811)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

Toolbar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Toolbar.Font (Page 6006)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Toolbar.Font (Page 6006)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Toolbar.Font (Page 6006)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Toolbar.Font (Page 6006)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Toolbar.Font (Page 6006)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Toolbar.Font (Page 6006)

Toolbar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Toolbar.Margin (Page 6010)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Toolbar.Margin (Page 6010)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Toolbar.Margin (Page 6010)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Toolbar.Margin (Page 6010)

Toolbar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

Toolbar.Padding (Page 6013)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

Toolbar.Padding (Page 6013)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

Toolbar.Padding (Page 6013)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

Toolbar.Padding (Page 6013)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

SystemDiagnosisControl.ToolBar (Page 6002)

SystemDiagnosisControl.Top

Description

The "Top" property specifies the value of the Y coordinate.

Type

Int32

Access

Read-write

Syntax

```
SystemDiagnosisControl.Top
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Visible

Description

The "Visible" property specifies whether the system diagnostics control is visible.

Type

Bool

Access

Read-write

Syntax

```
SystemDiagnosisControl.Visible
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.Width

Description

The "Width" property specifies the width.

Type

UInt32

Access

Read-write

Syntax

```
SystemDiagnosisControl.Width
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the system diagnostics control.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Type

HmiWindowFlag

Access

Read-write

Syntax

```
SystemDiagnosisControl.WindowFlags
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the system diagnostics control.

Syntax

```
SystemDiagnosisControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the system diagnostics control.

Syntax

```
SystemDiagnosisControl.FireCommand(commandId, custom)
```

Parameters

`commandId`

Type: Int32

ID of the element of the toolbar or information bar

`custom`

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.GoToPlc()

Description

The "GoToPlc" method navigates to the next PLC.

Syntax

```
SystemDiagnosisControl.GoToPlc()
```

Parameters

--

Return value

--

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
SystemDiagnosisControl.PropertyFlashing(propertyName, enable[,
value][, alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl_OnActivated()

Description

The "OnActivated" event occurs when a system diagnostics control receives focus:

- A system diagnostics control is selected via the configured tab sequence.
- A system diagnostics control that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
SystemDiagnosisControl_OnActivated(item)
```

Context

item

Type: Object

System diagnostics control where the event occurs.

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the system diagnostics control.

Syntax

```
SystemDiagnosisControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

System diagnostics control where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when the system diagnostics control loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
SystemDiagnosisControl_OnDeactivated(item)
```

Context

item

Type: Object

System diagnostics control where the event occurs.

See also

SystemDiagnosisControl (Page 5750)

SystemDiagnosisControl_OnInitialized()

Description

The "OnInitialized" event occurs when a system diagnostics control has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
SystemDiagnosisControl_OnInitialized(item)
```

Context

item

Type: Object

System diagnostics control where the event occurs.

See also

SystemDiagnosisControl (Page 5750)

Text

Description

The "Text" object represents a text box without frame and background.

Object type

HmiText

Properties

The "Text" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the text box.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height in the (Device Independent Unit).
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Layer**
  Returns the screen layer in which the text box is located.
- **Left**
  Specifies the value of the X coordinate in the (Device Independent Unit).
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the text box.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the text box is operable.

- **Parent**
  Returns the higher-level screen object (Parent container).
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the rotation angle of the text box in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the text box rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the text box is highlighted when in focus.
- **StyleItemClass**
  Specifies the style which is applied to the text box.
- **TabIndex**
  Returns the position of the text box in the tab sequence.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width in (Device Independent Unit).

Methods

The "Text" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the text box.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "Text" object has the following events:

- **OnActivated()**
  Occurs when a text box receives focus.
- **OnContextTapped()**
  Occurs when the text box is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a text box loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the text box is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the text box is in focus.
- **OnTapped()**
  Occurs when the text box is left-clicked or short-touched.

Text.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
Text.Authorization
```

See also

Text (Page 6025)

Text.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the text box.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
Text.CurrentQuality
```

See also

Text (Page 6025)

Text.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
Text.Enabled
```

See also

Text (Page 6025)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

Text (Page 6025)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 6029)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 6029)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 6029)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 6029)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 6029)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 6029)

Text.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

Text (Page 6025)

Text.Height

Description

The "Height" property specifies the height in the (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Text.Height
```

See also

Text (Page 6025)

Text.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered

- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Text.HorizontalTextAlignment
```

See also

Text (Page 6025)

Text.Layer

Description

The "Layer" property returns the screen layer in which the text box is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
Text.Layer
```

See also

Text (Page 6025)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

Text.Layer (Page 6034)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

Text.Layer (Page 6034)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

Text.Layer (Page 6034)

Text.Left

Description

The "Left" property sets the value of the X coordinate in the (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
Text.Left
```

See also

Text (Page 6025)

Text.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
Text.Margin
```

See also

Text (Page 6025)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

Text.Margin (Page 6037)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

Text.Margin (Page 6037)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

Text.Margin (Page 6037)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

Text.Margin (Page 6037)

Text.Name

Description

The "Name" property returns the name of the text box.

Type

String

Access

Read-only

Syntax

```
Text.Name
```

See also

Text (Page 6025)

Text.Opacity

Description

The "Opacity" property specifies the opacity. The "0" value indicates completely transparency.

Type

Float

Access

Read-write

Syntax

```
Text.Opacity
```

See also

Text (Page 6025)

Text.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
Text.Operability
```

See also

Text (Page 6025)

Text.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
Text.Parent
```

See also

Text (Page 6025)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

Text.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
Text.RequireExplicitUnlock
```

See also

Text (Page 6025)

Text.RotationAngle

Description

The "RotationAngle" property specifies the rotation angle of the text box in degrees.

Type

Int16

Access

Read-write

Syntax

```
Text.RotationAngle
```

See also

Text (Page 6025)
Text.RotationCenterPlacement (Page 6043)
Text.RotationCenterY (Page 6044)
Text.RotationCenterX (Page 6043)

Text.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the text box rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in the DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
Text.RotationCenterPlacement
```

See also

Text (Page 6025)

Text.RotationAngle (Page 6042)

Text.RotationCenterX (Page 6043)

Text.RotationCenterY (Page 6044)

Text.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point.

Type

Float

Access

Read-write

Syntax

```
Text.RotationCenterX
```

See also

Text (Page 6025)
Text.RotationAngle (Page 6042)
Text.RotationCenterPlacement (Page 6043)
Text.RotationCenterY (Page 6044)

Text.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point.

Type

Float

Access

Read-write

Syntax

```
Text.RotationCenterY
```

See also

Text (Page 6025)
Text.RotationAngle (Page 6042)
Text.RotationCenterPlacement (Page 6043)
Text.RotationCenterX (Page 6043)

Text.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the text box is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
Text.ShowFocusVisual
```

See also

Text (Page 6025)

Text.StyleItemClass

Description

The "StyleItemClass" property specifies the style which is applied to the text box.

Type

String

Access

Read-only

Syntax

```
Text.StyleItemClass
```

See also

Text (Page 6025)

Text.TabIndex

Description

The "TabIndex" property returns the position of the text box in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
Text.TabIndex
```

See also

Text (Page 6025)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

Text (Page 6025)

Text.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
Text.ToolTipText
```

See also

Text (Page 6025)

Text.Top

Description

The "Top" property specifies the value of the Y coordinate.

Type

Int32

Access

Read-write

Syntax

```
Text.Top
```

See also

Text (Page 6025)

Text.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Text.VerticalTextAlignment
```

See also

Text (Page 6025)

Text.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

Text (Page 6025)

Text.Width

Description

The "Width" property specifies the width in the (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
Text.Width
```

See also

Text (Page 6025)

Text.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the text box.

Syntax

```
Text.CheckAuthorization()
```

Parameters

-

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

Text (Page 6025)

Text.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
Text.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

Text (Page 6025)

Text_OnActivated()

Description

The "OnActivated" event occurs when a text box receives focus:

- A text box is selected via the configured tab sequence.
- A text box that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Text_OnActivated(item)
```

Context

item

Type: Object

Text box where the event occurs.

See also

Text (Page 6025)

Text_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A text box is right-clicked.
- A text box is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
Text_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Text box where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Text (Page 6025)

Text_OnDeactivated()

Description

The "OnDeactivated" event occurs when the text box loses focus because the operator has pressed the <TAB> key or executed another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
Text_OnDeactivated(item)
```

Context

item

Type: Object

Text box where the event occurs.

See also

Text (Page 6025)

Text_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the text box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Text_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Text box where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Text (Page 6025)

Text_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the text box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
Text_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Text box where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

Text (Page 6025)

Text_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A text box is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a text box has the focus.
- A text box is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
Text_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Text box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

Text (Page 6025)

TextBox

Description

The "TextBox" object represents a text box in runtime.

Object type

HmiTextBox

Properties

The "TextBox" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border thickness.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the text box.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Layer**
  Returns the layer of the screen in which the text box is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the text box.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border of the text box.
- **Parent**
  Returns the higher-level screen object.

- **ReadOnly**
  Specifies whether the text box is write-protected.
- **RenderingTemplate**
  Returns the name of the template from which the text box was created.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the angle of rotation in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the text box rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the text box is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the text box.
- **TabIndex**
  Returns the position of the text box in the tab sequence.
- **Text**
  Specifies the labeling of the text box.
- **TextTrimming**
  Specifies the type of text trimming if space is not sufficient.
- **TextWrapping**
  Specifies how text is wrapped if there is insufficient space.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "TextBox" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the text box.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "TextBox" object has the following events:

- **OnActivated()**
  Occurs when a text box receives focus.
- **OnContextTapped()**
  Occurs when a text box is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a text box loses focus.
- **OnKeyDown()**
  Occurs when a key is pressed while the text box is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the text box is in focus.
- **OnTapped()**
  Occurs when a text box is left-clicked or short-touched.

TextBox.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
TextBox.AlternateBackColor
```

See also

TextBox (Page 6057)

TextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
TextBox.AlternateBorderColor
```

See also

TextBox (Page 6057)

TextBox.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
TextBox.Authorization
```

See also

TextBox (Page 6057)

TextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TextBox.BackColor
```

See also

TextBox (Page 6057)

TextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
TextBox.BorderColor
```

See also

TextBox (Page 6057)

TextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
TextBox.BorderWidth
```

See also

TextBox (Page 6057)

TextBox.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the text box.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
TextBox.CurrentQuality
```

See also

TextBox (Page 6057)

TextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TextBox.Enabled
```

See also

TextBox (Page 6057)

TextBox.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
TextBox.Font
```

See also

TextBox (Page 6057)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

TextBox.Font (Page 6064)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

TextBox.Font (Page 6064)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

TextBox.Font (Page 6064)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

TextBox.Font (Page 6064)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

TextBox.Font (Page 6064)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

TextBox.Font (Page 6064)

TextBox.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
TextBox.ForeColor
```

See also

TextBox (Page 6057)

TextBox.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

Uint32

Access

Read-write

Syntax

```
TextBox.Height
```

See also

TextBox (Page 6057)

TextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the text alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
TextBox.HorizontalTextAlignment
```

See also

TextBox (Page 6057)

TextBox.Layer

Description

The "Layer" property returns the layer of the screen in which the text box is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
TextBox.Layer
```

See also

TextBox (Page 6057)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

TextBox.Layer (Page 6070)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

TextBox.Layer (Page 6070)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

TextBox.Layer (Page 6070)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

TextBox.Layer (Page 6070)

TextBox.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TextBox.Left
```

See also

TextBox (Page 6057)

TextBox.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
TextBox.Margin
```

See also

TextBox (Page 6057)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

TextBox.Margin (Page 6073)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

TextBox.Margin (Page 6073)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

TextBox.Margin (Page 6073)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

TextBox.Margin (Page 6073)

TextBox.Name

Description

The "Name" property returns the name of the text box.

Type

String

Access

Read-only

Syntax

```
TextBox.Name
```

See also

TextBox (Page 6057)

TextBox.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
TextBox.Opacity
```

See also

TextBox (Page 6057)

TextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
TextBox.Operability
```

See also

TextBox (Page 6057)

TextBox.Padding

Description

The "Padding" property specifies the distance of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
TextBox.Padding
```

See also

TextBox (Page 6057)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

TextBox.Padding (Page 6077)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

TextBox.Padding (Page 6077)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

TextBox.Padding (Page 6077)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

TextBox.Padding (Page 6077)

TextBox.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
TextBox.Parent
```

See also

TextBox (Page 6057)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

TextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
TextBox.ReadOnly
```

See also

TextBox (Page 6057)

TextBox.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the text box was created.

Type

String

Access

Read-only

Syntax

```
TextBox.RenderingTemplate
```

See also

TextBox (Page 6057)

TextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
TextBox.RequireExplicitUnlock
```

See also

TextBox (Page 6057)

TextBox.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
TextBox.RotationAngle
```

See also

TextBox (Page 6057)

TextBox.RotationCenterPlacement (Page 6082)

TextBox.RotationCenterX (Page 6083)

TextBox.RotationCenterY (Page 6083)

TextBox.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the text box rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
TextBox.RotationCenterPlacement
```

See also

TextBox (Page 6057)

TextBox.RotationCenterX (Page 6083)

TextBox.RotationCenterY (Page 6083)

TextBox.RotationAngle (Page 6081)

TextBox.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
TextBox.RotationCenterX
```

See also

TextBox (Page 6057)

TextBox.RotationCenterPlacement (Page 6082)

TextBox.RotationAngle (Page 6081)

TextBox.RotationCenterY (Page 6083)

TextBox.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
TextBox.RotationCenterY
```

See also

TextBox (Page 6057)

TextBox.RotationCenterPlacement (Page 6082)

TextBox.RotationAngle (Page 6081)

TextBox.RotationCenterX (Page 6083)

TextBox.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the text box is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
TextBox.ShowFocusVisual
```

See also

TextBox (Page 6057)

TextBox.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the text box.

Type

String

Access

Read-only

Syntax

```
TextBox.StyleItemClass
```

See also

TextBox (Page 6057)

TextBox.TabIndex

Description

The "TabIndex" property returns the position of the text box in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
TextBox.TabIndex
```

See also

TextBox (Page 6057)

TextBox.Text

Description

The "Text" property specifies the labeling of the text box.

Type

String

Access

Read-write

Syntax

```
TextBox.Text
```

See also

TextBox (Page 6057)

TextBox.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
TextBox.TextTrimming
```

See also

TextBox (Page 6057)

TextBox.TextWrapping

Description

The "TextWrapping" property specifies how text is wrapped if there is insufficient space.

Type

Int32, HmiTextWrapping

Specifies the text break:

- NoWrap (0): No text break
- WordWrap (1): Text break at the end of the line

Access

Read-write

Syntax

```
TextBox.TextWrapping
```

See also

TextBox (Page 6057)

TextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
TextBox.ToolTipText
```

See also

TextBox (Page 6057)

TextBox.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TextBox.Top
```

See also

TextBox (Page 6057)

TextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the text alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched. Can only be used with layout containers, Fallback behaves like Center

Access

Read-write

Syntax

```
TextBox.VerticalTextAlignment
```

See also

TextBox (Page 6057)

TextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
TextBox.Visible
```

See also

TextBox (Page 6057)

TextBox.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
TextBox.VisualizeQuality
```

See also

TextBox (Page 6057)

TextBox.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TextBox.Width
```

See also

TextBox (Page 6057)

TextBox.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the text box.

Syntax

```
TextBox.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

TextBox (Page 6057)

TextBox.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
TextBox.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

TextBox (Page 6057)

TextBox_OnActivated()

Description

The "OnActivated" event occurs when a text box receives focus:

- A text box is selected via the configured tab sequence.
- A text box that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
TextBox_OnActivated(item)
```

Context

item

Type: Object

Text box where the event occurs.

See also

TextBox (Page 6057)

TextBox_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A text box is right-clicked.
- A text box is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Syntax

```
TextBox_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Text box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

TextBox (Page 6057)

TextBox_OnDeactivated()

Description

The "OnDeactivated" event occurs when the text box loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
TextBox_OnDeactivated(item)
```

Context

item

Type: Object

Text box where the event occurs.

See also

TextBox (Page 6057)

TextBox_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the text box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
TextBox_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Text box where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

TextBox (Page 6057)

TextBox_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the text box is in focus.

Order

The events are triggered in the following order:

1. OnKeyDown
2. OnKeyUp

Syntax

```
TextBox_OnKeyUp(item, keyCode, modifiers)
```

Context

item

Type:

Text box where the event occurs.

keyCode

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

TextBox (Page 6057)

TextBox_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A text box is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a text box has the focus.
- A text box is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
3. OnTapped

Syntax

```
TextBox_OnTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Text box where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

TextBox (Page 6057)

ToggleSwitch

Description

The "ToggleSwitch" object represents a switch in runtime.

Object type

HmiToggleSwitch

Properties

The "ToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the border color.
- **BorderWidth**
  Specifies the border width.
- **Content**
  Specifies the display options for text and graphics.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the switch.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **Font**
  Specifies the font of the text.
- **ForeColor**
  Specifies the font color of the text.
- **Graphic**
  Specifies the graphic for the "not pressed" state.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the state of the switch.
- **Layer**
  Returns the screen layer in which the switch is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.

- **Name**
  Returns the name of the switch.
- **Opacity**
  Specifies the opacity.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border of the switch.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the switch was created.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **RotationAngle**
  Specifies the rotation angle in degrees.
- **RotationCenterPlacement**
  Specifies the reference point around which the switch rotates.
- **RotationCenterX**
  Specifies the X coordinate of the rotation point.
- **RotationCenterY**
  Specifies the Y coordinate of the rotation point.
- **ShowFocusVisual**
  Specifies whether the switch is highlighted when in focus.
- **StyleItemClass**
  Returns the style which is applied to the switch.
- **TabIndex**
  Returns the position of the switch in the tab sequence.
- **Text**
  Specifies the text for the "not pressed" state.
- **ToolTipText**
  Specifies the tooltip text.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the switch is visible.
- **VisualizeQuality**
  Specifies whether the connection quality of the process value is displayed.
- **Width**
  Specifies the width.

Methods

The "ToggleSwitch" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the switch.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "ToggleSwitch" object has the following events:

- **OnActivated()**
  Occurs when a switch receives focus.
- **OnContextTapped()**
  Occurs when a switch is right-clicked or long-touched.
- **OnDeactivated()**
  Occurs when a switch loses focus.
- **OnDown()**
  Occurs when the operator presses a switch.
- **OnKeyDown()**
  Occurs when a key is pressed while the switch is in focus.
- **OnKeyUp()**
  Occurs when a key is released while the switch is in focus.
- **OnStateChanged()**
  Occurs if the state of a switch changes.
- **OnTapped()**
  Occurs when a switch is left-clicked or short-touched.
- **OnUp()**
  Occurs when the operator resolves the pressure on a switch via the input device.

ToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.AlternateBackColor
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color which is displayed for line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.AlternateBorderColor
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic for the "pressed" state.

Type

String

Access

Read-write

Syntax

```
ToggleSwitch.AlternateGraphic
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text for the "pressed" state.

Type

String

Access

Read-write

Syntax

```
ToggleSwitch.AlternateText
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ToggleSwitch.Authorization
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.BackColor
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.BorderColor
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ToggleSwitch.BorderWidth
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Content

Description

The "Content" property specifies the display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ToggleSwitch.Content
```

See also

ToggleSwitch (Page 6098)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ToggleSwitch.Content (Page 6105)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ToggleSwitch.Content (Page 6105)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ToggleSwitch.Content (Page 6105)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ToggleSwitch.Content (Page 6105)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ToggleSwitch.Content (Page 6105)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ToggleSwitch.Content (Page 6105)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ToggleSwitch.Content (Page 6105)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ToggleSwitch.Content (Page 6105)

ToggleSwitch.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the switch.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
ToggleSwitch.CurrentQuality
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToggleSwitch.Enabled
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToggleSwitch.Font
```

See also

ToggleSwitch (Page 6098)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToggleSwitch.Font (Page 6112)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToggleSwitch.Font (Page 6112)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToggleSwitch.Font (Page 6112)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToggleSwitch.Font (Page 6112)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToggleSwitch.Font (Page 6112)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToggleSwitch.Font (Page 6112)

ToggleSwitch.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.ForeColor
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic for the "not pressed" state.

Type

String

Access

Read-write

Syntax

```
ToggleSwitch.Graphic
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.Height
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ToggleSwitch.HotKey
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the state of the switch:

- True: Pressed
- False: Not pressed

Type

Bool

Access

Read-write

Syntax

```
ToggleSwitch.IsAlternateState
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Layer

Description

The "Layer" property returns the layer of the screen in which the switch is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
ToggleSwitch.Layer
```

See also

ToggleSwitch (Page 6098)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

ToggleSwitch.Layer (Page 6118)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

ToggleSwitch.Layer (Page 6118)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

ToggleSwitch.Layer (Page 6118)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

ToggleSwitch.Layer (Page 6118)

ToggleSwitch.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ToggleSwitch.Left
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToggleSwitch.Margin
```

See also

ToggleSwitch (Page 6098)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToggleSwitch.Margin (Page 6121)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToggleSwitch.Margin (Page 6121)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToggleSwitch.Margin (Page 6121)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToggleSwitch.Margin (Page 6121)

ToggleSwitch.Name

Description

The "Name" property returns the name of the switch.

Type

String

Access

Read-only

Syntax

```
ToggleSwitch.Name
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Opacity

Description

The "Opacity" property specifies the opacity. The value "0" indicates completely transparent.

Type

Float

Access

Read-write

Syntax

```
ToggleSwitch.Opacity
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Operability

Description

The "Operability" property returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ToggleSwitch.Operability
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Padding

Description

The "Padding" property specifies the distance of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToggleSwitch.Padding
```

See also

ToggleSwitch (Page 6098)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToggleSwitch.Padding (Page 6125)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToggleSwitch.Padding (Page 6125)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToggleSwitch.Padding (Page 6125)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToggleSwitch.Padding (Page 6125)

ToggleSwitch.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
ToggleSwitch.Parent
```

See also

ToggleSwitch (Page 6098)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

ToggleSwitch.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the switch was created.

Type

String

Access

Read-only

Syntax

```
ToggleSwitch.RenderingTemplate
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ToggleSwitch.RequireExplicitUnlock
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.RotationAngle

Description

The "RotationAngle" property specifies the angle of rotation in degrees.

Type

Int16

Access

Read-write

Syntax

```
ToggleSwitch.RotationAngle
```

See also

ToggleSwitch (Page 6098)
ToggleSwitch.RotationCenterPlacement (Page 6130)
ToggleSwitch.RotationCenterX (Page 6130)
ToggleSwitch.RotationCenterY (Page 6131)

ToggleSwitch.RotationCenterPlacement

Description

The "RotationCenterPlacement" property sets the reference point around which the switch rotates.

Type

Int32, HmiRotationCenterPlacement

Specifies the reference point:

- AbsoluteFromCenter (0): Absolute distance from the object center in DIU (Device Independent Unit).
- NormedFromCenter (1): Relative distance from the center of the object to the surrounding object. The frame of the object is represented by the values "1" or "-1". The center of rotation can also lie outside the object.
- AbsoluteToContainer (2): Absolute distance from the screen origin (Top,Left: 0.0) in DIU (Device Independent Unit).

Access

Read-write

Syntax

```
ToggleSwitch.RotationCenterPlacement
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.RotationAngle (Page 6129)

ToggleSwitch.RotationCenterX (Page 6130)

ToggleSwitch.RotationCenterY (Page 6131)

ToggleSwitch.RotationCenterX

Description

The "RotationCenterX" property specifies the X coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
ToggleSwitch.RotationCenterX
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.RotationAngle (Page 6129)

ToggleSwitch.RotationCenterPlacement (Page 6130)

ToggleSwitch.RotationCenterY (Page 6131)

ToggleSwitch.RotationCenterY

Description

The "RotationCenterY" property specifies the Y coordinate of the rotation point. The value is the relative or absolute deviation from the center of the reference object.

Type

Float

Access

Read-write

Syntax

```
ToggleSwitch.RotationCenterY
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.RotationAngle (Page 6129)

ToggleSwitch.RotationCenterPlacement (Page 6130)

ToggleSwitch.RotationCenterX (Page 6130)

ToggleSwitch.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the switch is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
ToggleSwitch.ShowFocusVisual
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the switch.

Type

String

Access

Read-only

Syntax

```
ToggleSwitch.StyleItemClass
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.TabIndex

Description

The "TabIndex" property returns the position of the switch in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
ToggleSwitch.TabIndex
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Text

Description

The "Text" property specifies the text for the "not pressed" state.

Type

String

Access

Read-write

Syntax

```
ToggleSwitch.Text
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip.

Type

String

Access

Read-write

Syntax

```
ToggleSwitch.ToolTipText
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
ToggleSwitch.Top
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ToggleSwitch.Visible
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
ToggleSwitch.VisualizeQuality
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ToggleSwitch.Width
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the switch.

Syntax

```
ToggleSwitch.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

ToggleSwitch (Page 6098)

ToggleSwitch.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
ToggleSwitch.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnActivated()

Description

The "OnActivated" event occurs when a switch receives focus:

- A switch is selected via the configured tab sequence.
- A switch that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
ToggleSwitch_OnActivated(item)
```

Context

item

Type: Object

Switch where the event occurs.

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnContextTapped()

Description

The "OnContextTapped" event occurs with the following inputs of the operator:

- A switch is right-clicked.
- A switch is long-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnDown
2. OnUp
3. OnContextTapped

Syntax

```
ToggleSwitch_OnContextTapped(item, x, y, modifiers, trigger)
```

Context

item

Type: Object

Switch where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnDeactivated()

Description

The "OnDeactivated" event occurs when the switch loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
ToggleSwitch_OnDeactivated(item)
```

Context

item

Type: Object

Switch where the event occurs.

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnDown()

Description

The "OnDown" event occurs when the operator presses a switch:

- A switch is clicked with a mouse button.
- The <RETURN> or <SPACE> key is pressed when a switch has the focus.
- A switch is touched.

Order

The events are triggered in the following order:

1. OnDown
2. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
3. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
4. OnUp

Syntax

```
ToggleSwitch_OnDown(item, x, y, modifiers, trigger)
```

Context

Item

Type: Object

Switch where the event occurs.

x

Type: DInt

X-coordinate of the input point

y

Type: DInt

Y-coordinate of the input point

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

trigger

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnKeyDown()

Description

The "OnKeyDown" event occurs when a key is pressed while the switch is in focus. If the key is <RETURN> or <SPACE>, an "OnKeyDown" event is triggered before an "OnDown" event.

Order

The events are triggered in the following order:

1. OnDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyDown
3. OnKeyUp
4. OnUp (if triggered by the <RETURN> or <SPACE> key)

Syntax

```
ToggleSwitch_OnKeyDown(item, keyCode, modifiers)
```

Context

item

Type: Object

Switch where the event occurs.

keyCode

Type: UInt

Virtual key code of the key that triggered the event (for example "27" for ESC).

modifiers

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnKeyUp()

Description

The "OnKeyUp" event occurs when a key is released while the switch is in focus. If the key is <RETURN> or <SPACE>, an "OnUp" event is triggered after the "OnKeyUp" event.

Order

The events are triggered in the following order:

1. OnDown (if triggered by the <RETURN> or <SPACE> key)
2. OnKeyDown
3. OnKeyUp
4. OnUp (if triggered by the <RETURN> or <SPACE> key)

Syntax

```
ToggleSwitch_OnKeyUp(item, keyCode, modifiers)
```

Context

`item`

Type:

Switch where the event occurs.

`keyCode`

Type:

Virtual key code of the key that triggered the event (for example "27" for ESC).

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnStateChanged()

Description

The "OnStateChanged" event occurs when the state of a switch changes, for example, from "On" to "Off".

Syntax

```
ToggleSwitch_OnStateChanged(item, isAlternate)
```

Context

item

Type: Object

Switch where the event occurs.

isAlternate

Type: Bool

Specifies the state of the switch.

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnTapped()

Description

The "OnTapped" event occurs with the following inputs of the operator:

- A switch is left-clicked.
- The <RETURN> or <SPACE> key is pressed when a switch has the focus.
- A switch is short-touched.

Note

The length of the contact is defined by a limit value: 1000 ms if it is not specified by the operating system or web browser.

Order

The events are triggered in the following order:

1. OnDown
2. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
3. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
4. OnUp
5. OnTapped

Syntax

```
ToggleSwitch_OnTapped(item, x, y, modifiers, trigger)
```

Context

`item`

Type: Object

Switch where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ToggleSwitch (Page 6098)

ToggleSwitch_OnUp()

Description

The event "OnUp" occurs when the operator releases the pressure on a switch via the input device:

- The mouse button is released via a switch.
- The <RETURN> or <SPACE> key is released when a switch has the focus.
- The touch of a switch is canceled.
- A switch is exited while pressed.

This event does not occur as long as the operator keeps the switch pressed.

Order

The events are triggered in the following order:

1. OnDown
2. OnKeyDown (if triggered by the <RETURN> or <SPACE> key)
3. OnKeyUp (if triggered by the <RETURN> or <SPACE> key)
4. OnUp

Syntax

```
ToggleSwitch_OnUp(item, x, y, modifiers, trigger)
```

Context

`Item`

Type: Object

Switch where the event occurs.

`x`

Type: DInt

X-coordinate of the input point

`y`

Type: DInt

Y-coordinate of the input point

`modifiers`

Type: Int32, HmiKeyboardModifier

Help key that is pressed as you type:

- None (0): None
- Control (1): Control key
- Shift (2): Shift key
- Control + Shift (3): Control key + Shift key
- Alt (4): Alternate key
- Control + Alt (5): Control key + Alternate key
- Shift + Alt (6): Shift key + Alternate key
- Control + Shift + Alt (7): Control key + Shift key + Alternate key

`trigger`

Type: Int32, HmiEventTrigger

The event is triggered as follows:

- Unknown (0): Unknown
- Touch (1): Touch (touch device)
- Left (16): Left-click (mouse)
- Middle (17): Middle-click (mouse)
- Right (18): Right-click (mouse)
- Enter (256): Enter key (keyboard)
- Space (257): Space key (keyboard)
- Escape (258): Escape key (keyboard)

See also

ToggleSwitch (Page 6098)

TopLevelScreenWindow

Description

TopLevelScreenWindow (Page 1522)

TouchArea

Description

The "TouchArea" object represents a touch area in runtime.

Object type

HmiTouchArea

Properties

The "TouchArea" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Enabled**
  Specifies whether the touch area can be operated in runtime.
- **Height**
  Specifies the height.
- **Layer**
  Returns the layer of the screen in which the touch area is located.
- **Left**
  Specifies the value of the X coordinate.
- **Name**
  Returns the name of the touch area.
- **Operability**
  Returns whether the touch area is operable.
- **Parent**
  Returns the higher-level screen object.
- **RequireExplicitUnlock**
  Returns whether the touch area is only operable while the associated button is being pressed.

- **StyleItemClass**
  Returns the style which is applied to the touch area.
- **TabIndex**
  Returns the position of the touch area in the tab sequence.
- **Top**
  Specifies the value of the Y coordinate.
- **Visible**
  Specifies whether the touch area is visible.
- **Width**
  Specifies the width.

Methods

The "TouchArea" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the touch area.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "TouchArea" object has the following events:

- **OnGestureDetected()**
  Occurs when the operator performs a touch gesture.

TouchArea.Authorization

Description

The "Authorization" property returns the operator authorization.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
TouchArea.Authorization
```

See also

TouchArea (Page 6149)

TouchArea.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TouchArea.BackColor
```

See also

TouchArea (Page 6149)

TouchArea.Enabled

Description

The "Enabled" property specifies whether the touch area can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TouchArea.Enabled
```

See also

TouchArea (Page 6149)

TouchArea.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TouchArea.Height
```

See also

TouchArea (Page 6149)

TouchArea.Layer

Description

The "Layer" property returns the layer of the screen in which the touch area is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
TouchArea.Layer
```

See also

TouchArea (Page 6149)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

TouchArea.Layer (Page 6152)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

TouchArea.Layer (Page 6152)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

TouchArea.Layer (Page 6152)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

TouchArea.Layer (Page 6152)

TouchArea.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TouchArea.Left
```

See also

TouchArea (Page 6149)

TouchArea.Name

Description

The "Name" property returns the name of the touch area.

Type

String

Access

Read-only

Syntax

```
TouchArea.Name
```

See also

TouchArea (Page 6149)

TouchArea.Operability

Description

The "Operability" property returns whether the option button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
TouchArea.Operability
```

See also

TouchArea (Page 6149)

TouchArea.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
TouchArea.Parent
```

See also

TouchArea (Page 6149)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

TouchArea.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the touch area can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
TouchArea.RequireExplicitUnlock
```

See also

TouchArea (Page 6149)

TouchArea.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the touch area.

Type

String

Access

Read-only

Syntax

```
TouchArea.StyleItemClass
```

See also

TouchArea (Page 6149)

TouchArea.TabIndex

Description

The "TabIndex" property returns the position of the touch area in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
TouchArea.TabIndex
```

See also

TouchArea (Page 6149)

TouchArea.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TouchArea.Top
```

See also

TouchArea (Page 6149)

TouchArea.Visible

Description

The "Visible" property specifies whether the touch area is visible.

Type

Bool

Access

Read-write

Syntax

```
TouchArea.Visible
```

See also

TouchArea (Page 6149)

TouchArea.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TouchArea.Width
```

See also

TouchArea (Page 6149)

TouchArea.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the touch area.

Syntax

```
TouchArea.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

TouchArea (Page 6149)

TouchArea.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
TouchArea.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

TouchArea (Page 6149)

TouchArea_OnGestureDetected()

Description

The "OnGestureDetected" event occurs when the operator performs a touch gesture:

Syntax

```
TouchArea_OnGestureDetected(item, gesture)
```

Context

item

Type: Object

Touch area where the event occurs.

gesture

Type: Int32, HmiGesture

Touch gesture which is detected:

- Unknown (0): Unknown
- SwipeLeft (1): Swipe to the left
- SwipeRight (2): Swipe to the right
- SwipeUp (3): Swipe upwards
- SwipeDown (4): Swipe downwards

Example

Set the value of the "MyTag1" tag depending on the detected gesture:

Copy code

```
export function Touch_area_1_OnGestureDetected(item, gesture) {
// value of tag ‚MyTag1‘ will be set depending on the detected gesture
   if(gesture == UI.Enums.HmiGesture.SwipeRight)
   {
      UI.RootWindow.Screen = 'ScreenRight';
      let tag1 = Tags('tag1');
      tag1.Write(1); //write value '1234' to tag 'MyTag1'
   }
   if(gesture == UI.Enums.HmiGesture.SwipeLeft)
   {
      UI.RootWindow.Screen = 'ScreenLeft';
      let tag1 = Tags('tag1');
      tag1.Write(2); //write value '1234' to tag 'MyTag1'
   }
   if(gesture == UI.Enums.HmiGesture.SwipeUp)
   {
      UI.RootWindow.Screen = 'ScreenUp';
      let tag1 = Tags('tag1');
      tag1.Write(3); //write value '1234' to tag 'MyTag1'
   }
   if(gesture == UI.Enums.HmiGesture.SwipeDown)
   {
      UI.RootWindow.Screen = 'ScreenDown';
      let tag1 = Tags('tag1');
      tag1.Write(4); //write value '1234' to tag 'MyTag1'
   }
   if(gesture == UI.Enums.HmiGesture.Unknown)
   {
      let tag1 = Tags('tag1');
      tag1.Write(0); //write value '1234' to tag 'MyTag1'
   }
}
```

See also

TouchArea (Page 6149)

TrendCompanion

Description

The "TrendCompanion" object represents a value table in runtime.

Object type

HmiTrendCompanion

Properties

The "TrendCompanion" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the value table.
- **Enabled**
  Specifies whether the value table can be operated in runtime.
- **Height**
  Specifies the height in the (Device Independent Unit).
- **Icon**
  Specifies the icon of the value table.
- **Layer**
  Returns the screen layer in which the value table is located.
- **Left**
  Specifies the value of the X coordinate in the (Device Independent Unit).
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the value table.
- **Parent**
  Returns the higher-level screen object (Parent container).
- **RenderingTemplate**
  Returns the name of the template from which the value table was created.
- **ShowAlways**
  Specifies whether the value table can be closed.
- **ShowFocusVisual**
  Specifies whether the value table is highlighted when in focus.
- **SnapToSourceControl**
  Specifies whether the value table snaps to the window of the associated data source.
- **SourceTrendControl**
  Specifies the data source.
- **StatusBar**
  Specifies the information bar of the value table.
- **StyleItemClass**
  Specifies the style which is applied to the value table.
- **TabIndex**
  Returns the position of the value table in the tab sequence.

- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Specifies the toolbar of the value table.
- **Top**
  Specifies the value of the Y coordinate.
- **TrendCompanionMode**
  Specifies the window display of the value table.
- **TrendRulerView**
  Specifies the ruler window of the value table.
- **TrendStatisticAreaView**
  Specifies the view of the statistics area.
- **TrendStatisticResultView**
  Specifies the view of the statistics mode.
- **UseSourceControlBackColor**
  Specifies whether the background color of the value table is taken from the associated data source.
- **UseSourceControlTrendColors**
  Specifies whether the font color of the value table is taken from the associated data source.
- **Visible**
  Specifies whether the value table is visible.
- **Width**
  Specifies the width in (Device Independent Unit).
- **WindowFlags**
  Specifies the window configuration of the value table.

Methods

The "TrendCompanion" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the value table.
- **FireCommand()**
  Executes the command of an element of the toolbar or information bar of the value table.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "TrendCompanion" object has the following events:

- **OnActivated()**
  Occurs when the value table receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the value table.
- **OnDeactivated()**
  Occurs when the value table loses focus.
- **OnInitialized()**
  Occurs when the value table has been successfully initialized and the data connection to the PLC has been established.

TrendCompanion.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TrendCompanion.BackColor
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
TrendCompanion.Caption
```

See also

TrendCompanion (Page 6163)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

TrendCompanion.Caption (Page 6166)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 6167)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 6167)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 6167)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 6167)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 6167)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 6167)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

TrendCompanion.Caption (Page 6166)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

TrendCompanion.Caption (Page 6166)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

TrendCompanion.Caption (Page 6166)

TrendCompanion.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
TrendCompanion.CaptionColor
```

See also

TrendCompanion (Page 6163)

TrendCompanion.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the value table.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
TrendCompanion.CurrentQuality
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Enabled

Description

The "Enabled" property specifies whether the value table can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.Enabled
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TrendCompanion.Height
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Icon

Description

The "Icon" property specifies the icon of the value table.

Type

String

Access

Read-write

Syntax

```
TrendCompanion.Icon
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Layer

Description

The "Layer" property returns the screen layer in which the value table is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
TrendCompanion.Layer
```

See also

TrendCompanion (Page 6163)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

TrendCompanion.Layer (Page 6175)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

TrendCompanion.Layer (Page 6175)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

TrendCompanion.Layer (Page 6175)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

TrendCompanion.Layer (Page 6175)

TrendCompanion.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TrendCompanion.Left
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
TrendCompanion.Margin
```

See also

TrendCompanion (Page 6163)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

TrendCompanion.Margin (Page 6178)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

TrendCompanion.Margin (Page 6178)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

TrendCompanion.Margin (Page 6178)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

TrendCompanion.Margin (Page 6178)

TrendCompanion.Name

Description

The "Name" property returns the name of the value table.

Type

String

Access

Read-only

Syntax

```
TrendCompanion.Name
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
TrendCompanion.Parent
```

See also

TrendCompanion (Page 6163)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

TrendCompanion.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the value table was created.

Type

String

Access

Read-only

Syntax

```
TrendCompanion.RenderingTemplate
```

See also

TrendCompanion (Page 6163)

TrendCompanion.ShowAlways

Description

The "ShowAlways" property specifies whether the value table can be closed.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.ShowAlways
```

See also

TrendCompanion (Page 6163)

TrendCompanion.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the value table is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.ShowFocusVisual
```

See also

TrendCompanion (Page 6163)

TrendCompanion.SnapToSourceControl

Description

The "SnapToSourceControl" property specifies whether the value table snaps to the window of the associated data source.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.SnapToSourceControl
```

See also

TrendCompanion (Page 6163)

TrendCompanion.SourceTrendControl

Description

The "SourceTrendControl" property specifies the data source.

Type

Object, HmiTrendControlBase (Page 6374)

Access

Read-write

Syntax

```
TrendCompanion.SourceTrendControl
```

See also

TrendCompanion (Page 6163)

TrendControl (Page 6374)

TrendControl

Description

TrendControl (Page 6374)

TrendCompanion.StatusBar

Description

The "StatusBar" property specifies the information bar of the value table.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
TrendCompanion.StatusBar
```

See also

TrendCompanion (Page 6163)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

TrendCompanion.StatusBar (Page 6184)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 6185)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

TrendCompanion.StatusBar (Page 6184)

HmiControlBarElementCollection (Page 6185)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 6185)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6185)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6187)

See also

HmiControlBarElementCollection (Page 6185)

Control Bar Elements (Page 6187)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.

- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 6185)

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 6187)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 6187)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 6187)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 6187)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 6187)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 6193)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 6193)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 6193)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 6193)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 6193)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 6193)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 6193)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 6193)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 6187)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 6187)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 6187)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 6203)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 6203)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 6203)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 6203)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 6187)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 6187)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 6187)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 6187)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 6208)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 6208)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 6208)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 6208)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 6187)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 6187)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 6187)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 6185)

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 6213)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 6215)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 6215)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 6215)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 6215)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 6215)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 6215)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 6215)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 6215)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 6213)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 6225)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 6225)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 6225)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 6225)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 6213)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 6230)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 6230)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 6230)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 6230)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 6213)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 6213)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.

- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 6185)

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 6235)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 6242)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 6242)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 6242)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 6242)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 6235)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 6247)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 6247)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 6247)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 6247)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 6235)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 6235)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.

- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 6185)

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 6253)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 6259)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 6259)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 6259)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 6259)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 6253)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 6264)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 6264)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 6264)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 6264)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 6253)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 6253)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.

- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 6185)

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 6268)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 6275)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 6275)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 6275)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 6275)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 6268)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 6282)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 6282)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 6282)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 6282)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 6268)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 6268)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.

- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

See also

HmiControlBarElementCollection (Page 6185)

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 6289)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 6295)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 6295)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 6289)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 6306)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 6306)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 6306)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 6306)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 6289)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 6311)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 6311)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 6311)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 6311)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 6289)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 6289)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

TrendCompanion.StatusBar (Page 6184)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

TrendCompanion.StatusBar (Page 6184)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 6317)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 6317)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 6317)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 6317)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 6317)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 6317)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

TrendCompanion.StatusBar (Page 6184)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 6321)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 6321)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 6321)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 6321)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

TrendCompanion.StatusBar (Page 6184)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 6323)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 6323)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 6323)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 6323)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

TrendCompanion.StatusBar (Page 6184)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

TrendCompanion.StatusBar (Page 6184)

TrendCompanion.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the value table.

Type

String

Access

Read-only

Syntax

```
TrendCompanion.StyleItemClass
```

See also

TrendCompanion (Page 6163)

TrendCompanion.TabIndex

Description

The "TabIndex" property returns the position of the value table in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
TrendCompanion.TabIndex
```

See also

TrendCompanion (Page 6163)

TrendCompanion.TimeZone

Description

The "TimeZone" property specifies the time zone.

Type

HmiTimeZone

Access

Read-write

Syntax

```
TrendCompanion.TimeZone
```

See also

TrendCompanion (Page 6163)

TrendCompanion.ToolBar

Description

The "ToolBar" property specifies the toolbar of the value table.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
TrendCompanion.ToolBar
```

See also

TrendCompanion (Page 6163)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

TrendCompanion.ToolBar (Page 6328)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 6330)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

TrendCompanion.ToolBar (Page 6328)

HmiControlBarElementCollection (Page 6330)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 6329)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6330)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6332)

See also

HmiControlBarElementCollection (Page 6330)

Control Bar Elements (Page 6332)

Control Bar Elements

Description

Control Bar Elements (Page 6187)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

TrendCompanion.ToolBar (Page 6328)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

TrendCompanion.ToolBar (Page 6328)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 6332)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 6332)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 6332)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 6332)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 6332)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 6332)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

TrendCompanion.ToolBar (Page 6328)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 6336)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 6336)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 6336)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 6336)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

TrendCompanion.ToolBar (Page 6328)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 6339)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 6339)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 6339)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 6339)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

TrendCompanion.ToolBar (Page 6328)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

TrendCompanion.ToolBar (Page 6328)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

TrendCompanion.ToolBar (Page 6328)

TrendCompanion.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TrendCompanion.Top
```

See also

TrendCompanion (Page 6163)

TrendCompanion.TrendCompanionMode

Description

The "TrendCompanionMode" property specifies the window display of the value table.

Type

HmiTrendCompanionMode

Specifies the window display of the value table:

- Ruler (0): Ruler as reading aid
- StatisticArea (1): Statistics area
- StatisticResult (2): Statistics result

Access

Read-write

Syntax

```
TrendCompanion.TrendCompanionMode
```

See also

TrendCompanion (Page 6163)

TrendCompanion.TrendRulerView

Description

The "TrendRulerView" property specifies the ruler window of the value table.

Type

Object, HmiTrendColumnPart (Page 6344)

Access

Read-write

Syntax

```
TrendCompanion.TrendRulerView
```

See also

TrendCompanion (Page 6163)

TrendColumn (Page 6344)

TrendColumn

Description

The "TrendColumn" object represents a column in the trend view.

Object type

HmiTrendColumnPart

Properties

The "TrendColumn" object has the following properties:

- **AllowSort**
  Specifies whether column sorting is allowed.
- **BackColor**
  Specifies the background color.

- **Content**
  Specifies display options for text and graphics.
- **Enabled**
  Specifies whether the column can be operated in runtime.
- **ForeColor**
  Specifies the font color of the text.
- **Header**
  Specifies the properties of the column header.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumWidth**
  Specifies the minimum width.
- **Name**
  Returns the name of the column.
- **OutputFormat**
  Specifies the format for displaying values.
- **SortDirection**
  Specifies the sorting direction.
- **SortOrder**
  Specifies the sorting order.
- **TrendInfoBlock**
  Specifies the info block.
- **Visible**
  Specifies whether the column is visible.
- **Width**
  Specifies the width of the column in the DIU (Device Independent Unit).

Methods

--

See also

TrendCompanion.TrendRulerView (Page 6344)

TrendColumn.AllowSort

Description

The "AllowSort" property specifies whether the sorting of the column is permitted.

This property is ignored if the parent object has the "AllowSort" property set to "False".

Type

Bool

Access

Read-write

Syntax

```
TrendColumn.AllowSort
```

See also

TrendColumn (Page 6344)

TrendColumn.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TrendColumn.BackColor
```

See also

TrendColumn (Page 6344)

TrendColumn.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
TrendColumn.Content
```

See also

TrendColumn (Page 6344)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

TrendColumn.Content (Page 6346)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

TrendColumn.Content (Page 6346)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered

- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

TrendColumn.Content (Page 6346)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

TrendColumn.Content (Page 6346)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

TrendColumn.Content (Page 6346)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

TrendColumn.Content (Page 6346)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

TrendColumn.Content (Page 6346)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

TrendColumn.Content (Page 6346)

TrendColumn.Enabled

Description

The "Enabled" property specifies whether the column can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TrendColumn.Enabled
```

See also

TrendColumn (Page 6344)

TrendColumn.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
TrendColumn.ForeColor
```

See also

TrendColumn (Page 6344)

TrendColumn.Header

Description

The "Header" property specifies the properties of the column header.

Type

Object, HmiDataGridColumnHeaderPart

Access

Read-write

Syntax

```
TrendColumn.Header
```

See also

TrendColumn (Page 6344)

DataGridColumnHeader.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
DataGridColumnHeader.Content
```

See also

TrendColumn.Header (Page 6353)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

DataGridColumnHeader.Content (Page 6353)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

DataGridColumnHeader.Content (Page 6353)

DataGridColumnHeader.Graphic

Description

The "Graphic" property specifies the graphic of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Graphic
```

See also

TrendColumn.Header (Page 6353)

DataGridColumnHeader.Text

Description

The "Text" property specifies the label of the column header.

Type

String

Access

Read-write

Syntax

```
DataGridColumnHeader.Text
```

See also

TrendColumn.Header (Page 6353)

TrendColumn.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
TrendColumn.MaximumWidth
```

See also

TrendColumn (Page 6344)

TrendColumn.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
TrendColumn.MinimumWidth
```

See also

TrendColumn (Page 6344)

TrendColumn.Name

Description

The "Name" property returns the name of the column.

Type

String

Access

Read-only

Syntax

```
TrendColumn.Name
```

See also

TrendColumn (Page 6344)

TrendColumn.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying values.

Type

String

Access

Read-write

Syntax

```
TrendColumn.OutputFormat
```

See also

TrendColumn (Page 6344)

TrendColumn.SortDirection

Description

The "SortDirection" property specifies the direction of the sorting.

Type

Int32, HmiSortDirection

Specifies the sorting order.

- None (0): None
- Ascending (1): Ascending
- Descending (2): Descending

Access

Read-write

Syntax

```
TrendColumn.SortDirection
```

See also

TrendColumn (Page 6344)

TrendColumn.SortOrder

Description

The "SortOrder" property specifies the order of the sorting.

The sorting index starts at "1" (highest priority) in ascending order. Zero is ignored.

Type

UInt16

Access

Read-write

Syntax

```
TrendColumn.SortOrder
```

See also

TrendColumn (Page 6344)

TrendColumn.TrendInfoBlock

Description

The "TrendInfoBlock" property specifies the info block.

Type

Int32, HmiTrendInfoBlock

Specifies the info block:

- None (0): None
- Name (1): Name
- Index (2): Index
- Label (3): Labels
- Show (4): Show
- TagNameY (5): Tag name X
- TagNameX (6): Tag name Y
- YValue (7): Y value
- XValueOrTimestamp (8): X values or time stamp
- YValueLowerLimit (9): Y value low limit
- TimestampLowerLimit (10): Time stamp low limit
- YValueUpperLimit (11): Y value high limit
- TimestampUpperLimit (12): Time stamp low limit
- Minimum (13): Minimum
- MinimumTimestamp (14): Minimum time stamp
- Maximum (15): Maximum
- MaximumTimestamp (16): Maximum time stamp
- Average (17): Average
- StandardDeviation (18): Standard deviation
- Integral (19): Integral
- WeightedAverageValue (20): Weighted mean
- Duration (21): Duration
- NumberOfValues (22): Number of values
- AreaName (23): Area name

- AreaNameLL (24): Area name (LL)
- AreaNameHL (25): Area name (HL)
- Sum (32): Total

Access

Read-write

Syntax

```
TrendColumn.TrendInfoBlock
```

See also

TrendColumn (Page 6344)

TrendColumn.Visible

Description

The "Visible" property specifies whether the column is visible.

Type

Bool

Access

Read-write

Syntax

```
TrendColumn.Visible
```

See also

TrendColumn (Page 6344)

TrendColumn.Width

Description

The "Width" property specifies the width of the column in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TrendColumn.Width
```

See also

TrendColumn (Page 6344)

TrendCompanion.TrendStatisticAreaView

Description

The "TrendStatisticAreaView" property specifies the view of the statistics area.

Type

Object, HmiTrendColumnPart (Page 6365)

Access

Read-write

Syntax

```
TrendCompanion.TrendStatisticAreaView
```

See also

TrendCompanion (Page 6163)

TrendColumn (Page 6365)

TrendColumn

Description

TrendColumn (Page 6344)

TrendCompanion.TrendStatisticResultView

Description

The "TrendStatisticResultView" property specifies the view of the statistics mode.

Type

Object, HmiTrendColumnPart (Page 6366)

Access

Read-write

Syntax

```
TrendCompanion.TrendStatisticResultView
```

See also

TrendCompanion (Page 6163)

TrendColumn (Page 6366)

TrendColumn

Description

TrendColumn (Page 6344)

TrendCompanion.UseSourceControlBackColor

Description

The "UseSourceControlBackColor" property specifies whether the background color of the value table is taken from the associated data source.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.UseSourceControlBackColor
```

See also

TrendCompanion (Page 6163)

TrendCompanion.UseSourceControlTrendColors

Description

The "UseSourceControlTrendColors"property specifies whether the font color of the value table is taken from the associated data source.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.UseSourceControlTrendColors
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Visible

Description

The "Visible" property specifies whether the value table is visible.

Type

Bool

Access

Read-write

Syntax

```
TrendCompanion.Visible
```

See also

TrendCompanion (Page 6163)

TrendCompanion.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TrendCompanion.Width
```

See also

TrendCompanion (Page 6163)

TrendCompanion.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the value table.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized

- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
TrendCompanion.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

TrendCompanion (Page 6163)

TrendCompanion.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the value table.

Syntax

```
TrendCompanion.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

TrendCompanion (Page 6163)

TrendCompanion.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the value table.

Syntax

```
TrendCompanion.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

TrendCompanion (Page 6163)

TrendCompanion.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
TrendCompanion.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

TrendCompanion (Page 6163)

TrendCompanion_OnActivated()

Description

The "OnActivated" event occurs when a value table receives focus:

- A value table is selected via the configured tab sequence.
- A value table that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
TrendCompanion_OnActivated(item)
```

Context

`item`

Type: Object

Value table where the event occurs.

See also

TrendCompanion (Page 6163)

TrendCompanion_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the value table.

Syntax

```
TrendCompanion_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Value table where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

TrendCompanion (Page 6163)

TrendCompanion_OnDeactivated()

Description

The "OnDeactivated" event occurs when a value table loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

Syntax

```
TrendCompanion_OnDeactivated(item)
```

Context

item

Type: Object

Value table where the event occurs.

See also

TrendCompanion (Page 6163)

TrendCompanion_OnInitialized()

Description

The "OnInitialized" event occurs when a value table has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
TrendCompanion_OnInitialized(item)
```

Context

item

Type: Object

Value table where the event occurs.

See also

TrendCompanion (Page 6163)

TrendControl

Description

The "TrendControl" object represents a trend control of tag values from the current process or the archive in runtime.

Object type

HmiTrendControl

Properties

The "TrendControl" object has the following properties:

- **AreaSpacing**
  Specifies the distance between trend areas.
- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text to be displayed in the title bar.
- **CaptionColor**
  Specifies the color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the trend control.
- **Enabled**
  Specifies whether the trend control can be operated in runtime.
- **ExtendRulerToAxis**
  Specifies whether the ruler is extended into the axis.
- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the trend control.
- **Layer**
  Returns the screen layer in which the trend control is located.
- **Left**
  Specifies the value of the X coordinate.
- **Legend**
  Specifies the appearance of the legend.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the trend control.
- **Online**
  Specifies the start and stop of the trend control updating.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the trend control was created.
- **ShiftAxis**
  Specifies whether to swap the x axis and y axis of the trend control.

- **ShowFocusVisual**
  Specifies whether the trend control is highlighted when in focus.
- **ShowRuler**
  Specifies whether the ruler is displayed for determining a trend value.
- **ShowStatisticRulers**
  Specifies whether to display the two rulers for specifying the statistics area.
- **StatusBar**
  Sets the information bar of the trend control.
- **StyleItemClass**
  Returns the style which is applied to the trend control.
- **TabIndex**
  Returns the position of the trend control in the tab sequence.
- **TimeZone**
  Specifies the time zone.
- **ToolBar**
  Specifies the toolbar of the trend control.
- **Top**
  Specifies the value of the Y coordinate.
- **TrendAreas**
  Returns the trend areas of the trend control.
- **Visible**
  Specifies whether the trend control is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the trend control.

Methods

The "TrendControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the trend control.
- **FireCommand()**
  Executes the command of an element of the toolbar or information bar of the trend view.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "TrendControl" object has the following events:

- **OnActivated()**
  Occurs when a trend control receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the trend control.
- **OnDeactivated()**
  Occurs when a trend control loses focus.
- **OnInitialized()**
  Occurs when a trend control has been successfully initialized and the data connection to the PLC has been established.

TrendControl.AreaSpacing

Description

The "AreaSpacing" property specifies the spacing between trend areas.

Type

UInt16

Access

Read-write

Syntax

```
TrendControl.AreaSpacing
```

See also

TrendControl (Page 6374)

TrendControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TrendControl.BackColor
```

See also

TrendControl (Page 6374)

TrendControl.Caption

Description

The "Caption" property specifies the text to be displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
TrendControl.Caption
```

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

TrendControl.Caption (Page 6378)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 6378)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 6378)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 6378)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 6378)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 6378)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 6378)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

TrendControl.Caption (Page 6378)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

TrendControl.Caption (Page 6378)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

TrendControl.Caption (Page 6378)

TrendControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
TrendControl.CaptionColor
```

See also

TrendControl (Page 6374)

TrendControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the trend view.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the quality of the value is worse than usual.
- Good (4): Usable, quality of the value is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
TrendControl.CurrentQuality
```

See also

TrendControl (Page 6374)

TrendControl.Enabled

Description

The "Enabled" property specifies whether the trend view can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.Enabled
```

See also

TrendControl (Page 6374)

TrendControl.ExtendRulerToAxis

Description

The "ExtendRulerToAxis" property specifies whether the ruler is extended to the axis.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.ExtendRulerToAxis
```

See also

TrendControl (Page 6374)

TrendControl.Height

Description

The "Height" property specifies the height in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TrendControl.Height
```

See also

TrendControl (Page 6374)

TrendControl.Icon

Description

The "Icon" property specifies the icon of the trend view.

Type

String

Access

Read-write

Syntax

```
TrendControl.Icon
```

See also

TrendControl (Page 6374)

TrendControl.Layer

Description

The "Layer" property returns the layer of the screen in which the trend view is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
TrendControl.Layer
```

See also

TrendControl (Page 6374)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

TrendControl.Layer (Page 6387)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

TrendControl.Layer (Page 6387)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

TrendControl.Layer (Page 6387)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

TrendControl.Layer (Page 6387)

TrendControl.Left

Description

The "Left" property sets the value of the X coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TrendControl.Left
```

See also

TrendControl (Page 6374)

TrendControl.Legend

Description

The "Legend" property specifies the appearance of the legend ("Legend" object).

Type

Object, HmiLegendPart

Access

Read-write

Syntax

```
TrendControl.Legend
```

See also

TrendControl (Page 6374)

Legend.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Legend.ForeColor
```

See also

TrendControl.Legend (Page 6390)

Legend.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Legend.Font
```

See also

TrendControl.Legend (Page 6390)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Legend.Font (Page 6391)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Legend.Font (Page 6391)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Legend.Font (Page 6391)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Legend.Font (Page 6391)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Legend.Font (Page 6391)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Legend.Font (Page 6391)

Legend.Visible

Description

The "Visible" property specifies whether the legend is visible.

Type

Bool

Access

Read-write

Syntax

```
Legend.Visible
```

See also

TrendControl.Legend (Page 6390)

TrendControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
TrendControl.Margin
```

See also

TrendControl (Page 6374)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

TrendControl.Margin (Page 6395)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

TrendControl.Margin (Page 6395)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

TrendControl.Margin (Page 6395)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

TrendControl.Margin (Page 6395)

TrendControl.Name

Description

The "Name" property returns the name of the trend view.

Type

String

Access

Read-only

Syntax

```
TrendControl.Name
```

See also

TrendControl (Page 6374)

TrendControl.Online

Description

The "Online" property specifies the start and stop of the updating of the trend view.

- True: Online. The trend view is updated with new values.
- False: Offline. No new values are added to the trend view.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.Online
```

See also

TrendControl (Page 6374)

TrendControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
TrendControl.Parent
```

See also

TrendControl (Page 6374)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

TrendControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the trend view was created.

Type

String

Access

Read-only

Syntax

```
TrendControl.RenderingTemplate
```

See also

TrendControl (Page 6374)

TrendControl.ShiftAxis

Description

The "ShiftAxis" property specifies whether to swap the x axis and y axis of the trend view.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.ShiftAxis
```

See also

TrendControl (Page 6374)

TrendControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the trend view is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.ShowFocusVisual
```

See also

TrendControl (Page 6374)

TrendControl.ShowRuler

Description

The "ShowRuler" property specifies whether the ruler is displayed for determining a trend value.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.ShowRuler
```

See also

TrendControl (Page 6374)

TrendControl.ShowStatisticRulers

Description

The "ShowStatisticRulers" property specifies whether to display the two rulers for specifying the statistics area.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.ShowStatisticRulers
```

See also

TrendControl (Page 6374)

TrendControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the trend view.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
TrendControl.StatusBar
```

See also

TrendControl (Page 6374)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

TrendControl.StatusBar (Page 6402)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 6403)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

HmiControlBarElementCollection (Page 6403)

TrendControl.StatusBar (Page 6402)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 6403)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6403)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6421)

See also

HmiControlBarElementCollection (Page 6403)

Control Bar Elements (Page 6421)

Control Bar Elements

Description

Control Bar Elements (Page 6421)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

TrendControl.StatusBar (Page 6402)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

TrendControl.StatusBar (Page 6402)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 6406)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 6406)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 6406)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 6406)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 6406)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 6406)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

TrendControl.StatusBar (Page 6402)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 6410)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 6410)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 6410)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 6410)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

TrendControl.StatusBar (Page 6402)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 6412)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 6412)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 6412)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 6412)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

TrendControl.StatusBar (Page 6402)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

TrendControl.StatusBar (Page 6402)

TrendControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the trend view.

Type

String

Access

Read-only

Syntax

```
TrendControl.StyleItemClass
```

See also

TrendControl (Page 6374)

TrendControl.TabIndex

Description

The "TabIndex" property returns the position of the trend view in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
TrendControl.TabIndex
```

See also

TrendControl (Page 6374)

TrendControl.TimeZone

Description

The "TimeZone" property specifies the time zone.

Positive numbers according to the Microsoft time zone index value specification, negative numbers from CHROM (-1 = RH local)

Type

HmiTimeZone

Values according to the Microsoft Time Zone Index Value specification. "-1" stands for local time.

Access

Read-write

Syntax

```
TrendControl.TimeZone
```

See also

TrendControl (Page 6374)

TrendControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the trend view.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
TrendControl.ToolBar
```

See also

TrendControl (Page 6374)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

TrendControl.ToolBar (Page 6417)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 6419)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

HmiControlBarElementCollection (Page 6419)

TrendControl.ToolBar (Page 6417)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 6418)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6419)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6421)

See also

HmiControlBarElementCollection (Page 6419)

Control Bar Elements (Page 6421)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.

- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 6421)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 6421)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 6421)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 6421)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 6421)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 6426)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.

- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 6426)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 6426)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 6426)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 6426)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 6426)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 6426)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 6426)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 6421)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 6421)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 6421)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 6437)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 6437)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 6437)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 6437)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 6421)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 6421)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 6421)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 6421)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 6442)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 6442)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 6442)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 6442)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 6421)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 6421)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 6421)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.

- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 6447)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 6447)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 6449)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 6449)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 6449)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 6449)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 6449)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 6449)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 6449)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 6449)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page

- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 6447)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 6447)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 6459)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 6459)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 6459)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 6459)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 6447)

ControlBarButton.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 6447)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 6464)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 6464)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 6464)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 6464)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 6447)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 6447)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 6469)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 6476)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 6476)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 6476)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 6476)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 6469)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 6481)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 6481)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 6481)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 6481)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 6469)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 6469)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 6486)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 6492)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 6492)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 6492)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 6492)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 6486)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 6497)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 6497)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 6497)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 6497)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 6486)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 6486)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.

- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 6502)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 6511)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 6511)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 6511)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 6511)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 6502)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 6516)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 6516)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 6516)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 6516)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 6502)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 6502)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".

- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 6522)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 6529)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 6529)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 6522)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 6540)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 6540)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 6540)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 6540)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 6522)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 6545)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 6545)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 6545)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 6545)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 6522)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 6522)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

TrendControl.ToolBar (Page 6417)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

TrendControl.ToolBar (Page 6417)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 6551)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 6551)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 6551)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 6551)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 6551)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 6551)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

TrendControl.ToolBar (Page 6417)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 6554)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 6554)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 6554)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 6554)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

TrendControl.ToolBar (Page 6417)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 6557)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 6557)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 6557)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 6557)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

TrendControl.ToolBar (Page 6417)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

TrendControl.ToolBar (Page 6417)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

TrendControl.ToolBar (Page 6417)

TrendControl.Top

Description

The "Top" property sets the value of the Y coordinate in DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
TrendControl.Top
```

See also

TrendControl (Page 6374)

TrendControl.TrendAreas

Description

The "TrendAreas" property returns the trend areas ("TrendArea" objects) of the trend control.

Type

Object, HmiTrendAreaCollection (Page 6562)

Access

Read-only

Syntax

```
TrendControl.TrendAreas
```

See also

TrendControl (Page 6374)

HmiTrendAreaCollection (Page 6562)

HmiTrendAreaCollection

Description

The "HmiTrendAreaCollection" object is a list of all trend areas ("TrendArea" objects) of the trend view.

Use

The "HmiTrendAreaCollection" object is a list and can be counted and enumerated. You can access the "HmiTrendAreaCollection" list using the index or the tag name.

Object type

HmiTrendAreaCollection

Properties

The "HmiTrendAreaCollection" object has the following properties:

- **Count**
  Returns the number of trend areas of the "HmiTrendAreaCollection" list.

Methods

The "HmiTrendAreaCollection" object has the following methods:

- **Item()**
  Returns a trend area of the "HmiTrendAreaCollection" list.

See also

TrendControl.TrendAreas (Page 6561)

HmiTrendAreaCollection.Count

Description

The "Count" property returns the number of trend areas in the "HmiTrendAreaCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiTrendAreaCollection.Count
```

See also

HmiTrendAreaCollection (Page 6562)

HmiTrendAreaCollection.Item()

Description

The "Item" method returns a trend area of the "HmiTrendAreaCollection" list.

Syntax

```
HmiTrendAreaCollection[.Item](HmiTrendAreaName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiTrendAreaCollection" object.

Parameters

`HmiTrendAreaName`

Type: String

Name of the trend area

Return value

Object, HmiTrendAreaPart (Page 6564)

See also

HmiTrendAreaCollection (Page 6562)

TrendArea (Page 6564)

TrendArea

Description

The "TrendArea" object represents a trend view of the trend display.

Object type

HmiTrendAreaPart

Properties

The "TrendArea" object has the following properties:

- **BackColor**
  Specifies the background color.
- **BottomTimeAxes**
  Returns the lower time axes of the trend area.
- **GridLines**
  Specifies the grid lines of the trend area.
- **LeftValueAxes**
  Returns the left value axes of the trend area.
- **MajorGridLinesColor**
  Specifies the color of the main grid lines.
- **MinorGridLinesColor**
  Specifies the color of the auxiliary grid lines.
- **Name**
  Returns the name of the trend area.
- **RightValueAxes**
  Returns the right value axes of the trend area.
- **Ruler**
  Specifies the appearance of the ruler to determine the trend value.
- **SelectedTrend**
  Specifies the selected trend of the trend area.
- **SizeFactor**
  Specifies the scaling factor of the trend area relative to its height.

- **StatisticRulers**
  Specifies the two rulers for specifying the statistics area.
- **TopTimeAxes**
  Returns the upper time axes of the trend area.
- **Trends**
  Returns the trends of the trend area.
- **Visible**
  Specifies whether the trend area is visible.

Methods

--

TrendArea.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
TrendArea.BackColor
```

See also

TrendArea (Page 6564)

TrendArea.BottomTimeAxes

Description

The "BottomTimeAxes" property returns the lower time axes of the trend area.

Type

Object, HmiTimeAxisCollection (Page 6566)

Access

Read-only

Syntax

```
TrendArea.BottomTimeAxes
```

See also

TrendArea (Page 6564)

HmiTimeAxisCollection (Page 6566)

HmiTimeAxisCollection

Description

The "HmiTimeAxisCollection" object is a list of all time axes ("TimeAxis" objects) of the trend area.

Use

The "HmiTimeAxisCollection" object is a list and can be counted and enumerated. You can access the "HmiTimeAxisCollection" list using the index or the tag name.

Object type

HmiTimeAxisCollection

Properties

The "HmiTimeAxisCollection" object has the following properties:

- **Count**
  Returns the number of time axes of the "HmiTimeAxisCollection" list.

Methods

The "HmiTimeAxisCollection" object has the following methods:

- **Item()**
  Returns a time axis of the "HmiTimeAxisCollection" list.

See also

TrendArea.BottomTimeAxes (Page 6565)

HmiTimeAxisCollection.Count

Description

The "Count" property returns the number of time axes in the "HmiTimeAxisCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiTimeAxisCollection.Count
```

See also

HmiTimeAxisCollection (Page 6566)

HmiTimeAxisCollection.Item()

Description

The "Item" method returns a time axis of the "HmiTimeAxisCollection" list.

Syntax

```
HmiTimeAxisCollection[.Item](HmiTimeAxisName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiTimeAxisCollection" object.

Parameters

`HmiTimeAxisName`

Type: String

Name of the time axis

Return value

Object, HmiTimeAxisPart (Page 6568)

See also

HmiTimeAxisCollection (Page 6566)

TimeAxis (Page 6568)

TimeAxis

Description

The "TimeAxis" object represents a time axis of the trend area.

Object type

HmiTimeAxisPart

Properties

The "TimeAxis" object has the following properties:

- **AlwaysShowRecent**
  Specifies whether the area with the youngest value is always displayed.
- **AutoScaling**
  Specifies whether the automatic scaling is activated.
- **AxisColor**
  Specifies the color of the time axis.
- **BeginTime**
  Specifies the date and time for the start time of the time range.
- **DisplayName**
  Specifies the display name of the time axis.
- **EndTime**
  Specifies the date and time for the end time of the time range.
- **LabelColor**
  Specifies the color of the axis labeling.
- **LabelFont**
  Specifies the font of the axis labeling.
- **OutputFormat**
  Specifies the format for displaying the time axis values.
- **PointCount**
  Specifies the number of measurement points from the start time.
- **RangeType**
  Specifies the type of time range.

- **ScaleMode**
  Specifies the type of scaling.
- **TickColor**
  Specifies the color of the tick marks.
- **TimeRangeBase**
  Specifies the basis of the time range.
- **TimeRangeFactor**
  Specifies the factor of the time base for defining the time range.
- **Visible**
  Specifies whether the time axis is visible.

Methods

--

TimeAxis.AlwaysShowRecent

Description

The "AlwaysShowRecent" property specifies whether the area with the most recent value is always displayed.

Type

Bool

Access

Read-write

Syntax

```
TimeAxis.AlwaysShowRecent
```

See also

TimeAxis (Page 6568)

TimeAxis.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
TimeAxis.AutoScaling
```

See also

TimeAxis (Page 6568)

TimeAxis.AxisColor

Description

The "AxisColor" property specifies the color of the time axis.

Type

UInt32

Access

Read-write

Syntax

```
TimeAxis.BackColor
```

See also

TimeAxis (Page 6568)

TimeAxis.BeginTime

Description

The "BeginTime" property specifies the date and time for the start time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
TimeAxis.BeginValue
```

See also

TimeAxis (Page 6568)

TimeAxis.DisplayName

Description

The "DisplayName" property specifies the display name of the time axis.

Type

String

Access

Read-write

Syntax

```
TimeAxis.DisplayName
```

See also

TimeAxis (Page 6568)

TimeAxis.EndTime

Description

The "EndTime" property specifies the date and time for the end time of the time range.

Type

DateTime

Access

Read-write

Syntax

```
TimeAxis.EndValue
```

See also

TimeAxis (Page 6568)

TimeAxis.LabelColor

Description

The "LabelColor" property specifies the color of the axis labeling.

Type

UInt32

Access

Read-write

Syntax

```
TimeAxis.LabelColor
```

See also

TimeAxis (Page 6568)

TimeAxis.LabelFont

Description

The "LabelFont" property specifies the font of the axis labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
TimeAxis.LabelFont
```

See also

TimeAxis (Page 6568)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

TimeAxis.LabelFont (Page 6572)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

TimeAxis.LabelFont (Page 6572)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

TimeAxis.LabelFont (Page 6572)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

TimeAxis.LabelFont (Page 6572)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

TimeAxis.LabelFont (Page 6572)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal

- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

TimeAxis.LabelFont (Page 6572)

TimeAxis.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the axis values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
TimeAxis.OutputFormat
```

See also

TimeAxis (Page 6568)

TimeAxis.PointCount

Description

The "PointCount" property specifies the number of measurement points from the start time.

Type

Int32

Access

Read-write

Syntax

```
TimeAxis.PointCount
```

See also

TimeAxis (Page 6568)

TimeAxis.RangeType

Description

The "RangeType" property specifies the type of time range.

Type

Int32, HmiTimeRangeType

Specifies the time range:

- TimeRange (0): Any time range
- FromBeginToEnd (1): Total time range
- PointCount (2): Number of measurement points

Access

Read-write

Syntax

```
TimeAxis.RangeType
```

See also

TimeAxis (Page 6568)

TimeAxis.ScaleMode

Description

The "ScaleMode" property specifies the type of scaling.

Type

Int32, HmiScaleMode

Specifies the scaling:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
TimeAxis.ScaleMode
```

See also

TimeAxis (Page 6568)

TimeAxis.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
TimeAxis.TickColor
```

See also

TimeAxis (Page 6568)

TimeAxis.TimeRangeBase

Description

The "TimeRangeBase" property specifies the base of the time range.

Type

Int32, HmiTimeRangeBase

Specifies a time range:

- Undefined (0): Not defined
- Millisecond (1): Millisecond
- Second (2): Second
- Minute (3): Minute
- Hour (4): Hour
- Day (5): Day
- Month (6): Month
- Year (7): Year

Access

Read-write

Syntax

```
TimeAxis.TimeRangeBase
```

See also

TimeAxis (Page 6568)

TimeAxis.TimeRangeFactor

Description

The "TimeRangeFactor" property specifies the factor of the time base for defining the time range.

Type

Int32

Access

Read-write

Syntax

```
TimeAxis.TimeRangeFactor
```

See also

TimeAxis (Page 6568)

TimeAxis.Visible

Description

The "Visible" property specifies whether the time axis is visible.

Type

Bool

Access

Read-write

Syntax

```
TimeAxis.Visible
```

See also

TimeAxis (Page 6568)

TrendArea.GridLines

Description

The "GridLines" property specifies the grid lines of the trend area.

Type

Int32, HmiGridLine

Specifies the display of the grid lines:

- None (0): None
- VerticalMajor (1): Vertical, coarse
- HorizontalMajor (2): Horizontal, coarse
- VerticalMinor (4): Vertical, fine
- HorizontalMinor (8): Horizontal, fine

Access

Read-write

Syntax

```
TrendArea.GridLines
```

See also

TrendArea (Page 6564)

TrendArea.LeftValueAxes

Description

The "LeftValueAxes" property returns the left value axes of the trend area.

Type

Object, HmiYValueAxisCollection (Page 6582)

Access

Read-only

Syntax

```
TrendArea.LeftValueAxes
```

See also

TrendArea (Page 6564)

HmiYValueAxisCollection (Page 6582)

HmiYValueAxisCollection

Description

The "HmiYValueAxisCollection" object is a list of all value axes ("YValueAxis" objects) of the trend area.

Use

The "HmiYValueAxisCollection" object is a list and can be counted and enumerated. You can access the "HmiYValueAxisCollection" list using the index or the tag name.

Object type

HmiYValueAxisCollection

Properties

The "HmiYValueAxisCollection" object has the following properties:

- **Count**
  Returns the number of value axes in the "HmiYValueAxisCollection" list.

Methods

The "HmiYValueAxisCollection" object has the following methods:

- **Item()**
  Returns a value axis of the "HmiYValueAxisCollection" list.

See also

TrendArea.LeftValueAxes (Page 6581)

HmiYValueAxisCollection.Count

Description

The "Count" property returns the number of value axes in the "HmiYValueAxisCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiYValueAxisCollection.Count
```

See also

HmiYValueAxisCollection (Page 6582)

HmiYValueAxisCollection.Item()

Description

The "Item" method returns a value axis of the "HmiYValueAxisCollection" list.

Syntax

```
HmiYValueAxisCollection[.Item](HmiYValueAxisName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiYValueAxisCollection" object.

Parameters

`HmiYValueAxisName`

Type: String

Name of the value axis

Return value

Object, HmiYValueAxisPart (Page 6583)

See also

HmiYValueAxisCollection (Page 6582)

YValueAxis (Page 6583)

YValueAxis

Description

The "YValueAxis" object represents a value axis of the trend area.

Object type

HmiYValueAxisPart

Properties

The "YValueAxis" object has the following properties:

- **ApplyScalingEntries**
  Specifies whether the user scaling of the axis section is applied.
- **AutoRange**
  Specifies whether automatic determination of the value range is activated by the minimum and maximum value of the trend.
- **AutoScaling**
  Specifies whether the automatic scaling is activated.
- **AxisColor**
  Specifies the color of the value axis.
- **BeginValue**
  Specifies the start of a value range or value range section.
- **DisplayName**
  Specifies the display name of the value axis.
- **DivisionCount**
  Specifies the number of main units with subdivisions.
- **EndValue**
  Specifies the end of a value range or value range section.
- **HelpLines**
  Returns the appearance of the help lines.
- **LabelColor**
  Specifies the color of the axis labeling.
- **LabelFont**
  Specifies the font of the axis labeling.
- **LargeTickLabelingStep**
  Specifies the interval at which scale sections are labeled.
- **MeasurementUnit**
  Returns the displayed unit.
- **MeasurementUnitType**
  Specifies the display format of the unit.
- **OutputFormat**
  Specifies the format for displaying the axis values.
- **ScalingEntries**
  Returns the specification of the user scaling of the axis sections.
- **ScaleMode**
  Specifies the scale mode:

- **ScalingType**
  Specifies the scaling.
- **ShowScalingDisplayNames**
  Specifies whether the display names of the user scaling are used.
- **SubDivisionCount**
  Specifies the number of divisions of the main units.
- **TickColor**
  Specifies the color of the tick marks.
- **Visible**
  Specifies whether the value axis is visible.

Methods

--

YValueAxis.ApplyScalingEntries

Description

The "ApplyScalingEntries" property specifies whether the user scaling of the axis sections is applied.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.ApplyScalingEntries
```

See also

YValueAxis (Page 6583)

YValueAxis.AutoRange

Description

The "AutoRange" property specifies whether automatic determination of the value range is activated by the minimum and maximum value of the trend.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.AutoRange
```

See also

YValueAxis (Page 6583)

YValueAxis.AutoScaling

Description

The "AutoScaling" property specifies whether automatic scaling is activated.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.AutoScaling
```

See also

YValueAxis (Page 6583)

YValueAxis.AxisColor

Description

The "AxisColor" property specifies the color of the value axis.

Type

UInt32

Access

Read-write

Syntax

```
YValueAxis.BackColor
```

See also

YValueAxis (Page 6583)

YValueAxis.BeginValue

Description

The "BeginValue" property specifies the start of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
YValueAxis.BeginValue
```

See also

YValueAxis (Page 6583)

YValueAxis.DisplayName

Description

The "DisplayName" property specifies the display name of the value axis.

Type

String

Access

Read-write

Syntax

```
YValueAxis.DisplayName
```

See also

YValueAxis (Page 6583)

YValueAxis.DivisionCount

Description

The "DivisionCount" property specifies the number of main units with subdivisions. To this purpose the automatic scaling must be switched off.

Type

Int32

Access

Read-write

Syntax

```
YValueAxis.DivisionCount
```

See also

YValueAxis (Page 6583)

YValueAxis.EndValue

Description

The "EndValue" property specifies the end of a value range or value range section.

Type

Float

Access

Read-write

Syntax

```
YValueAxis.EndValue
```

See also

YValueAxis (Page 6583)

YValueAxis.HelpLines

Description

The "HelpLines" property returns the appearance of the help lines.

Type

Object, HmiHelpLineCollection (Page 6589)

Access

Read-only

Syntax

```
YValueAxis.HelpLines
```

See also

YValueAxis (Page 6583)

HmiHelpLineCollection (Page 6589)

HmiHelpLineCollection

Description

The "HmiHelpLineCollection" object is a list of all help lines ("HelpLine" objects).

Use

The "HmiHelpLineCollection" object is a list and can be counted and enumerated. You can access the "HmiHelpLineCollection" list using the index or the tag name.

Object type

HmiHelpLineCollection

Properties

The "HmiHelpLineCollection" object has the following properties:

- **Count**
  Returns the number of help lines of the "HmiHelpLineCollection" list.

Methods

The "HmiHelpLineCollection" object has the following methods:

- **Item()**
  Returns a help line of the "HmiHelpLineCollection" list.

See also

YValueAxis.HelpLines (Page 6589)

HmiHelpLineCollection.Count

Description

The "Count" property returns the number of help lines in the "HmiHelpLineCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiHelpLineCollection.Count
```

See also

HmiHelpLineCollection (Page 6589)

HmiHelpLineCollection.Item()

Description

The "Item" method returns help line of the "HmiHelpLineCollection" list.

Syntax

```
HmiHelpLineCollection[.Item](HmiHelpLineName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiHelpLineCollection" object.

Parameters

HmiHelpLineName

Type: String

Name of the help line

Return value

Object, HmiHelpLinePart (Page 6591)

See also

HmiHelpLineCollection (Page 6589)

HelpLine (Page 6591)

HelpLine

Description

The "HelpLine" object represents a help line of the value axis.

Object type

HmiHelpLinePart

Properties

The "HelpLine" object has the following properties:

- **Value**
  Specifies the value of the help line.
- **Visible**
  Specifies whether the help line is visible.

Methods

--

HelpLine.Value

Description

The "Value" property sets the value of the help line.

Type

Float

Access

Read-write

Syntax

```
HelpLine.Value
```

See also

HelpLine (Page 6591)

HelpLine.Visible

Description

The "Visible" property specifies whether the help line is visible.

Type

Bool

Access

Read-write

Syntax

```
HelpLine.Visible
```

See also

HelpLine (Page 6591)

YValueAxis.LabelColor

Description

The "LabelColor" property specifies the color of the axis labeling.

Type

UInt32

Access

Read-write

Syntax

```
YValueAxis.LabelColor
```

See also

YValueAxis (Page 6583)

YValueAxis.LabelFont

Description

The "LabelFont" property specifies the font of the axis labeling.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
YValueAxis.LabelFont
```

See also

YValueAxis (Page 6583)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

YValueAxis.LabelFont (Page 6593)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

YValueAxis.LabelFont (Page 6593)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

YValueAxis.LabelFont (Page 6593)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

YValueAxis.LabelFont (Page 6593)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

YValueAxis.LabelFont (Page 6593)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

YValueAxis.LabelFont (Page 6593)

YValueAxis.LargeTickLabelingStep

Description

The "LargeTickLabelingStep" property specifies the interval at which scale sections are labeled.

Type

UInt8

Access

Read-write

Syntax

```
YValueAxis.LargeTickLabelingStep
```

See also

YValueAxis (Page 6583)

YValueAxis.MeasurementUnit

Description

The "MeasurementUnit" property returns the displayed unit.

Type

String

Access

Read-only

Syntax

```
YValueAxis.MeasurementUnit
```

See also

YValueAxis (Page 6583)

YValueAxis.MeasurementUnitType

Description

The "MeasurementUnitType" property specifies the display format of the unit.

Type

Int32, HmiMeasurementUnit

Specifies the display format:

- None (0): No unit
- Name (1): Unit name, for example "kilogram"
- Symbol (2): Unit, for example "kg"

Access

Read-write

Syntax

```
YValueAxis.MeasurementUnitType
```

See also

YValueAxis (Page 6583)

YValueAxis.OutputFormat

Description

The "OutputFormat" property specifies the format for displaying the axis values, e.g. "{0000}" for a 4-digit integer with leading zeros.

Type

String

Access

Read-write

Syntax

```
YValueAxis.OutputFormat
```

See also

YValueAxis (Page 6583)

YValueAxis.ScalingEntries

Description

The "ScalingEntries" property returns the user scaling of the axes sections.

Type

Object, HmiScalingEntryCollection (Page 6600)

Access

Read-only

Syntax

```
YValueAxis.ScalingEntries
```

See also

YValueAxis (Page 6583)

HmiScalingEntryCollection (Page 6600)

HmiScalingEntryCollection

Description

The "HmiScalingEntryCollection" object is a list of all user-defined axis sections ("ScalingEntry" objects).

Use

The "HmiScalingEntryCollection" object is a list and can be counted and enumerated. You can access the "HmiScalingEntryCollection" list using the index or the tag name.

Object type

HmiHelpLineCollection

Properties

The "HmiScalingEntryCollection" object has the following properties:

- **Count**
  Returns the number of user-defined axis sections of the "HmiScalingEntryCollection" list.

Methods

The "HmiScalingEntryCollection" object has the following methods:

- **Item()**
  Returns a user-defined axis section of the "HmiScalingEntryCollection" list.

See also

YValueAxis.ScalingEntries (Page 6599)

HmiScalingEntryCollection.Count

Description

The "Count" property returns the number of the user-defined axis sections in the "HmiScalingEntryCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiScalingEntryCollection.Count
```

See also

HmiScalingEntryCollection (Page 6600)

HmiScalingEntryCollection.Item()

Description

The "Item" method returns a user-defined axis section of the "HmiScalingEntryCollection" list.

Syntax

```
HmiScalingEntryCollection[.Item](HmiScalingEntryName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiScalingEntryCollection" object.

Parameters

`HmiScalingEntryName`

Type: String

Name of the user-defined axis section

Return value

Object, HmiScalingEntryPart (Page 6602)

See also

HmiScalingEntryCollection (Page 6600)

Scaling Entry (Page 6602)

Scaling Entry

Description

The "ScalingEntry" object represents a user-defined axis section of the value axis.

Object type

HmiScalingEntryPart

Properties

The "YValueAxis" object has the following properties:

- **BeginValue**
  Specifies the start of a value range section.
- **BeginValueTarget**
  Specifies the scaled value for the start of a value range section.
- **DisplayName**
  Specifies the display name of an axis section.
- **EndValue**
  Specifies the end of a value range section.
- **EndValueTarget**
  Specifies the scaled value for the end of a value range section.

Methods

--

ScalingEntry.BeginValue

Description

The "BeginValue" property specifies the start of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.BeginValue
```

See also

Scaling Entry (Page 6602)

ScalingEntry.BeginValueTarget

Description

The"BeginValueTarget" property specifies the scaled value for the start of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.BeginValueTarget
```

See also

Scaling Entry (Page 6602)

ScalingEntry.DisplayName

Description

The "DisplayName" property specifies the display name of an axis section.

Type

String

Access

Read-write

Syntax

```
ScalingEntry.DisplayName
```

See also

Scaling Entry (Page 6602)

ScalingEntry.EndValue

Description

The "EndValue" property specifies the end of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.EndValue
```

See also

Scaling Entry (Page 6602)

ScalingEntry.EndValueTarget

Description

The"EndValueTarget" property specifies the scaled value for the end of a value range section.

Type

Float

Access

Read-write

Syntax

```
ScalingEntry.EndValueTarget
```

See also

Scaling Entry (Page 6602)

YValueAxis.ScaleMode

Description

The "ScaleMode" property specifies the scale mode.

Type

Int32, HmiScaleMode

Specifies the scale mode:

- None (0): None
- Labels (1): Labels
- Ticks (2): Tick marks

Access

Read-write

Syntax

```
YValueAxis.ScaleMode
```

See also

YValueAxis (Page 6583)

YValueAxis.ScalingType

Description

The "ScalingType" property specifies the scaling.

Type

Int32, HmiScalingType

Specifies the scaling:

- Linear (0): Linear
- Logarithmic (1): Logarithmic
- NegativeLogarithmic (2): Negative logarithmic
- Tangent (4): Tangential
- Quadratic (5): Square
- Cubic (6): Cubic

Access

Read-write

Syntax

```
YValueAxis.ScalingType
```

See also

YValueAxis (Page 6583)

YValueAxis.ShowScalingDisplayNames

Description

The "ShowScalingDisplayNames" property specifies whether the display names of the user scaling are used.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.ShowScalingDisplayNames
```

See also

YValueAxis (Page 6583)

YValueAxis.SubDivisionCount

Description

The "SubDivisionCount" property specifies the number of subdivisions of the main units.

Type

Int32

Access

Read-write

Syntax

```
YValueAxis.SubDivisionCount
```

See also

YValueAxis (Page 6583)

YValueAxis.TickColor

Description

The "TickColor" property specifies the color of the tick marks.

Type

UInt32

Access

Read-write

Syntax

```
YValueAxis.TickColor
```

See also

YValueAxis (Page 6583)

YValueAxis.Visible

Description

The "Visible" property specifies whether the value axis is visible.

Type

Bool

Access

Read-write

Syntax

```
YValueAxis.Visible
```

See also

YValueAxis (Page 6583)

TrendArea.MajorGridLinesColor

Description

The "MajorGridLinesColor" property specifies the color of the main grid lines.

Type

UInt32

Access

Read-write

Syntax

```
TrendArea.MajorGridLinesColor
```

See also

TrendArea (Page 6564)

TrendArea.MinorGridLinesColor

Description

The "MinorGridLinesColor" specifies the color of the auxiliary grid lines.

Type

UInt32

Access

Read-write

Syntax

```
TrendArea.MinorGridLinesColor
```

See also

TrendArea (Page 6564)

TrendArea.Name

Description

The "Name" property returns the name of the trend view.

Type

String

Access

Read-only

Syntax

```
TrendArea.Name
```

See also

TrendArea (Page 6564)

TrendArea.RightValueAxes

Description

The "RightValueAxes" property returns the right value axes of the trend area.

Type

Object, HmiYValueAxisCollection (Page 6610)

Access

Read-only

Syntax

```
TrendArea.RightValueAxes
```

See also

TrendArea (Page 6564)

HmiYValueAxisCollection (Page 6610)

HmiYValueAxisCollection

Description

The "HmiYValueAxisCollection" object is a list of all value axes ("YValueAxis" objects) of the trend area.

Use

The "HmiYValueAxisCollection" object is a list and can be counted and enumerated. You can access the "HmiYValueAxisCollection" list using the index or the tag name.

Object type

HmiYValueAxisCollection

Properties

The "HmiYValueAxisCollection" object has the following properties:

- **Count**
  Returns the number of value axes in the "HmiYValueAxisCollection" list.

Methods

The "HmiYValueAxisCollection" object has the following methods:

- **Item()**
  Returns a value axis of the "HmiYValueAxisCollection" list.

See also

TrendArea.RightValueAxes (Page 6609)

HmiYValueAxisCollection.Count

Description

The "Count" property returns the number of value axes in the "HmiYValueAxisCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiYValueAxisCollection.Count
```

See also

HmiYValueAxisCollection (Page 6610)

HmiYValueAxisCollection.Item()

Description

The "Item" method returns a value axis of the "HmiYValueAxisCollection" list.

Syntax

```
HmiYValueAxisCollection[.Item](HmiYValueAxisName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiYValueAxisCollection" object.

Parameters

`HmiYValueAxisName`

Type: String

Name of the value axis

Return value

Object, HmiYValueAxisPart (Page 6583)

See also

HmiYValueAxisCollection (Page 6610)

YValueAxis (Page 6583)

YValueAxis

Description

YValueAxis (Page 6583)

TrendArea.Ruler

Description

The "Ruler" property specifies the appearance of the ruler for determining the trend value.

Type

Object, HmiRulerPart

Access

Read-write

Syntax

```
TrendArea.Ruler
```

See also

TrendArea (Page 6564)

Ruler.Color

Description

The "Color" property specifies the color of the ruler.

Type

UInt32

Access

Read-write

Syntax

```
Ruler.Color
```

See also

TrendArea.Ruler (Page 6612)

Ruler.Width

Description

The "Width" property specifies the width of the ruler.

Type

UInt32

Access

Read-write

Syntax

```
Ruler.Width
```

See also

TrendArea.Ruler (Page 6612)

TrendArea.SelectedTrend

Description

The "SelectedTrend" property specifies the selected trend of the trend area.

Type

Object, HmiTrendPart (Page 6621)

Access

Read-write

Syntax

```
TrendArea.SelectedTrend
```

See also

TrendArea (Page 6564)

Trend (Page 6621)

Trend

Description

Trend (Page 6621)

TrendArea.SizeFactor

Description

The "SizeFactor" property specifies the scaling factor of the trend area relative to its height.

Type

UInt16

Access

Read-write

Syntax

```
TrendArea.SizeFactor
```

See also

TrendArea (Page 6564)

TrendArea.StatisticRulers

Description

The "StatisticRulers" property specifies the two rulers for specifying the statistics area.

Type

Object, HmiRulerPart

Access

Read-write

Syntax

```
TrendArea.StatisticRulers
```

See also

TrendArea (Page 6564)

Ruler.Color

Description

The "Color" property specifies the color of the ruler.

Type

UInt32

Access

Read-write

Syntax

```
Ruler.Color
```

See also

TrendArea.StatisticRulers (Page 6615)

Ruler.Width

Description

The "Width" property specifies the width of the ruler.

Type

UInt32

Access

Read-write

Syntax

```
Ruler.Width
```

See also

TrendArea.StatisticRulers (Page 6615)

TrendArea.TopTimeAxes

Description

The "TopTimeAxes" property returns the upper value axes of the trend area.

Type

Object, HmiTimeAxisCollection (Page 6617)

Access

Read-only

Syntax

```
TrendArea.TopTimeAxes
```

See also

TrendArea (Page 6564)

HmiTimeAxisCollection (Page 6617)

HmiTimeAxisCollection

Description

The "HmiTimeAxisCollection" object is a list of all time axes ("TimeAxis" objects) of the trend area.

Use

The "HmiTimeAxisCollection" object is a list and can be counted and enumerated. You can access the "HmiTimeAxisCollection" list using the index or the tag name.

Object type

HmiTimeAxisCollection

Properties

The "HmiTimeAxisCollection" object has the following properties:

- **Count**
  Returns the number of time axes of the "HmiTimeAxisCollection" list.

Methods

The "HmiTimeAxisCollection" object has the following methods:

- **Item()**
  Returns a time axis of the "HmiTimeAxisCollection" list.

See also

TrendArea.TopTimeAxes (Page 6616)

HmiTimeAxisCollection.Count

Description

The "Count" property returns the number of time axes in the "HmiTimeAxisCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiTimeAxisCollection.Count
```

See also

HmiTimeAxisCollection (Page 6617)

HmiTimeAxisCollection.Item()

Description

The "Item" method returns a time axis of the "HmiTimeAxisCollection" list.

Syntax

```
HmiTimeAxisCollection[.Item](HmiTimeAxisName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiTimeAxisCollection" object.

Parameters

`HmiTimeAxisName`

Type: String

Name of the time axis

Return value

Object, HmiTimeAxisPart (Page 6568)

See also

HmiTimeAxisCollection (Page 6617)

TimeAxis (Page 6568)

TimeAxis

Description

TimeAxis (Page 6568)

TrendArea.Trends

Description

The "Trends" property returns the trends ("Trend" objects) of the trend area.

Type

Object, HmiTrendCollection (Page 6619)

Access

Read-only

Syntax

```
TrendArea.Trends
```

See also

TrendArea (Page 6564)

HmiTrendCollection (Page 6619)

HmiTrendCollection

Description

The "HmiTrendCollection" object is a list of all trends ("Trend" objects) of the trend view.

Use

The "HmiTrendCollection" object is a list and can be counted and enumerated. You can access the "HmiTrendCollection" list using the index or the tag name.

Object type

HmiTrendCollection

Properties

The "HmiTrendCollection" object has the following properties:

- **Count**
  Returns the number of trends of the "HmiTrendCollection" list.

Methods

The "HmiTrendCollection" object has the following methods:

- **Item()**
  Returns a trend of the "HmiTrendCollection" list.

See also

TrendArea.Trends (Page 6619)

HmiTrendCollection.Count

Description

The "Count" property returns the number of curves in the "HmiTrendCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiTrendCollection.Count
```

See also

HmiTrendCollection (Page 6619)

HmiTrendCollection.Item()

Description

The "Item" method returns a trend of the "HmiTrendCollection" list.

Syntax

```
HmiTrendCollection[.Item](HmiTrendName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiTrendCollection" object.

Parameters

`HmiTrendName`

Type: String

Trend name

Return value

Object, HmiTrendPart (Page 6621)

See also

HmiTrendCollection (Page 6619)

Trend (Page 6621)

Trend

Description

The "Trend" object represents a trend of the trend area.

Object type

HmiTrendPart

Properties

The "Trend" object has the following properties:

- **AggregationMode**
  Specifies the compression of values for logging tags.
- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **BackColor**
  Specifies the background color.
- **BackFillPattern**
  Specifies the pattern of the background or the fill.
- **DashType**
  Specifies the stroke style of the trend.
- **DataSourceY**
  Specifies the tag for the data source of the value axis.
- **DisplayName**
  Specifies the display name of the trend.

- **LineColor**
  Specifies the line color.
- **LineWidth**
  Specifies the line thickness.
- **MarkerColor**
  Specifies the color of the trend points.
- **MarkerDimension**
  Specifies the width of the trend points.
- **MarkerGraphic**
  Specifies a graphic object as a trend point.
- **MarkerType**
  Specifies the type of the trend points.
- **QualityVisualization**
  Specifies the colors for values of a specific quality.
- **ShowLoggedDataImmediately**
  Specifies which logged values are displayed.
- **Thresholds**
  Returns the list of all limit values of the trend.
- **TimeAxis**
  References a time axis of the trend.
- **TrendMode**
  Specifies the type of trend view.
- **Visible**
  Specifies whether the trend is visible.
- **YValueAxis**
  References a value axis of the trend.

Methods

--

Trend.AggregationMode

Description

The "AggregationMode" property specifies the compression of values for logging tags.

Type

Int32, HmiAggregationMode

Specifies the type of compression:

- None (0): None
- TimeAverageStepped (1): Average over time displayed in steps
- MinMax (2): Lowest and highest value from defined period

Access

Read-write

Syntax

```
Trend.AggregationMode
```

See also

Trend (Page 6621)

Trend.AlternateBackColor

Description

The "AlternateBackColor" specifies the second color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
Trend.AlternateBackColor
```

See also

Trend (Page 6621)

Trend.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
Trend.BackColor
```

See also

Trend (Page 6621)

Trend.BackFillPattern

Description

The "BackFillPattern" property specifies the pattern of the background or the fill.

Type

Int32, HmiFillPattern

Specifies the filling:

- Solid (0): Solid
- Transparent (65536): Transparent
- Horizontal (131072): Horizontal
- Vertical (131073): Vertical
- ForwardDiagonal (131074): Forward diagonal stripe
- BackwardDiagonal (131075): Backward diagonal stripe
- Cross (131076): Cross
- DiagonalCross (131077): Diagonal cross
- GradientHorizontal (1048576): Horizontal gradient
- GradientVertical (1048577): Vertical gradient
- GradientForwardDiagonal (1048578): Gradient forward diagonal
- GradientBackwardDiagonal (1048579): Gradient backward diagonal
- GradientHorizontalTricolor (1048832): Horizontal tricolor gradient
- GradientVerticalTricolor (1048833): Vertical tricolor gradient

- GradientForwardDiagonalTricolor (1048834): Forward diagonal tricolor gradient
- GradientBackwardDiagonalTricolor (1048835): Backward diagonal tricolor gradient

Access

Read-write

Syntax

```
Trend.BackFillPattern
```

See also

Trend (Page 6621)

Trend.DashType

Description

The "DashType" property specifies the stroke style type of the trend.

Type

Int32, HmiDashType

Specifies the stroke style:

- Solid (0): Solid
- Dash(1): Dashed
- Dot (2): Dotted
- DashDot (3): Dash-dot
- DashDotDot (4): Dash-dot-dot

Access

Read-write

Syntax

```
Trend.DashType
```

See also

Trend (Page 6621)

Trend.DataSourceY

Description

The "DataSourceY" property specifies the tag for the data source of the values axis.

Type

Object, HmiDataSourcePart

Access

Read-write

Syntax

```
Trend.DataSourceX
```

See also

Trend (Page 6621)

DataSource.Source

Description

The "Source" property specifies the data source, e.g. a tag or logging tag.

Type

String

Access

Read-write

Syntax

```
DataSource.Source
```

See also

Trend.DataSourceY (Page 6626)

DataSource.VisualizeQuality

Description

The "VisualizeQuality" property specifies whether the connection quality of the process value is displayed.

Type

Bool

Access

Read-write

Syntax

```
DataSource.VisualizeQuality
```

See also

Trend.DataSourceY (Page 6626)

Trend.DisplayName

Description

The "DisplayName" property specifies the display name of the trend.

Type

String

Access

Read-write

Syntax

```
Trend.DisplayName
```

See also

Trend (Page 6621)

Trend.LineColor

Description

The "LineColor" property specifies the line color.

Type

UInt32

Access

Read-write

Syntax

```
Trend.LineColor
```

See also

Trend (Page 6621)

Trend.LineWidth

Description

The "LineWidth" property specifies the line thickness.

Type

UInt8

Access

Read-write

Syntax

```
Trend.LineWidth
```

See also

Trend (Page 6621)

Trend.MarkerColor

Description

The "MarkerColor" property specifies the color of the trend points.

Type

UInt32

Access

Read-write

Syntax

```
Trend.MarkerColor
```

See also

Trend (Page 6621)

Trend.MarkerDimension

Description

The "MarkerDimension" property specifies the width of the trend points.

Type

UInt32

Access

Read-write

Syntax

```
Trend.MarkerDimension
```

See also

Trend (Page 6621)

Trend.MarkerGraphic

Description

The "MarkerGraphic" property specifies a graphic object as a trend point.

Type

String

Access

Read-write

Syntax

```
Trend.MarkerGraphic
```

See also

Trend (Page 6621)

Trend.MarkerType

Description

The "MarkerType" property specifies the type of trend points.

Type

Int32, HmiMarkerType

Specifies the type of the trend points:

- None (0): None
- Points (1): Dots
- Square (2): Square
- Circle (3): Circles
- Graphic (4): Graphic

Access

Read-write

Syntax

```
Trend.MarkerType
```

See also

Trend (Page 6621)

Trend.QualityVisualization

Description

The "QualityVisualization" property specifies the colors for values of a specific quality.

Type

Object, HmiQualityPart

Access

Read-write

Syntax

```
Trend.QualityVisualization
```

See also

Trend (Page 6621)

Quality.BadColor

Description

The "BadColor" property specifies the color for values of the quality "Bad". Values of this quality cannot be used.

Type

UInt32

Access

Read-write

Syntax

```
Quality.BadColor
```

See also

Trend.QualityVisualization (Page 6631)

Quality.UncertainColor

Description

The "UncertainColor" property specifies the color for values of the quality "Uncertain". The quality of this level's values is worse than usual. It might still be possible to use the values, however.

Type

UInt32

Access

Read-write

Syntax

```
Quality.UncertainColor
```

See also

Trend.QualityVisualization (Page 6631)

Quality.Visible

Description

The "Visible" property specifies whether the colors are visible.

Type

Bool

Access

Read-write

Syntax

```
Quality.Visible
```

See also

Trend.QualityVisualization (Page 6631)

Trend.ShowLoggedDataImmediately

Description

The "ShowLoggedDataImmediately" property specifies which logged values are displayed:

- True: Entire visible range
- False: Only from current time

Type

Bool

Access

Read-write

Syntax

```
Trend.ShowLoggedDataImmediately
```

See also

Trend (Page 6621)

Trend.Thresholds

Description

The "Thresholds" property returns the list of all limit values ("Threshold" objects) of the trend.

Type

Object, HmiThresholdCollection (Page 6634)

Access

Read-only

Syntax

```
Trend.Thresholds
```

See also

Trend (Page 6621)

HmiThresholdCollection (Page 6634)

HmiThresholdCollection

Description

The "HmiThresholdCollection" object is a list of all limit values ("Threshold" objects).

Use

The "HmiThresholdCollection" object is a list and can be counted and enumerated. You can access the "HmiThresholdCollection" list using the index or the tag name.

Object type

HmiThresholdCollection

Properties

The "HmiThresholdCollection" object has the following properties:

- **Count**
  Returns the number of limit values of the "HmiThresholdCollection" list.

Methods

The "HmiThresholdCollection" object has the following methods:

- **Item()**
  Returns a limit value of the "HmiThresholdCollection" list.

See also

Trend.Thresholds (Page 6633)

HmiThresholdCollection.Count

Description

The "Count" property returns the number of limit values in the "HmiThresholdCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiThresholdCollection.Count
```

See also

HmiThresholdCollection (Page 6634)

HmiThresholdCollection.Item()

Description

The "Item" method returns a limit value of the "HmiThresholdCollection" list.

Syntax

```
HmiThresholdCollection[.Item](HmiThresholdName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiThresholdCollection" object.

Parameters

`HmiThresholdName`

Type: String

Name of the limit value

Return value

Object, HmiThresholdPart (Page 6636)

See also

HmiThresholdCollection (Page 6634)

Threshold (Page 6636)

Threshold

Description

The "Threshold" object represents a limit value.

Object type

HmiThresholdPart

Properties

The "Threshold" object has the following properties:

- **Color**
  Specifies the color of the limit value.
- **DisplayName**
  Specifies the display name of the limit value.
- **Name**
  Specifies the name of the limit value.
- **ThresholdMode**
  Specifies the type of limit value.
- **Value**
  Returns the limit value.

Methods

--

Threshold.Color

Description

The "Color" property specifies the color of the limit value.

Type

UInt32

Access

Read-write

Syntax

```
Threshold.Color
```

See also

Threshold (Page 6636)

Threshold.DisplayName

Description

The "DisplayName" property specifies the display name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.DisplayName
```

See also

Threshold (Page 6636)

Threshold.Name

Description

The "Name" property specifies the name of the limit value.

Type

String

Access

Read-write

Syntax

```
Threshold.Name
```

See also

Threshold (Page 6636)

Threshold.ThresholdMode

Description

The "ThresholdMode" property specifies the type of limit value.

Type

Int32, HmiThresholdMode

Specifies the threshold value:

- Undefined (0): Undefined
- Upper (1): Upper threshold
- Lower (2): Lower threshold
- Normal (3): Normal threshold
- Minimum (4): Minimum threshold
- Maximum (5): Maximum threshold

Access

Read-write

Syntax

```
Threshold.ThresholdMode
```

See also

Threshold (Page 6636)

Threshold.Value

Description

The "Value" property returns the limit value of the tag.

Type

Float

Access

Read-only

Syntax

```
Threshold.Value
```

See also

Threshold (Page 6636)

Trend.TimeAxis

Description

The "TimeAxis" property references a time axis of the trend.

Type

Object, HmiTimeAxisPart (Page 6568)

Access

Read-write

Syntax

```
Trend.TimeAxis
```

See also

Trend (Page 6621)

TimeAxis (Page 6568)

TimeAxis

Description

TimeAxis (Page 6568)

Trend.TrendMode

Description

The "TrendMode" property specifies the type of trend view.

Type

Int32, HmiTrendMode

Specifies the trend type:

- Points (0): Dots
- Interpolated (1): Interpolated
- Stepped (2): Levels
- Bar (3): Bar
- Value (4): Values

Access

Read-write

Syntax

```
Trend.TrendMode
```

See also

Trend (Page 6621)

Trend.Visible

Description

The "Visible" property specifies whether the trend is visible.

Type

Bool

Access

Read-write

Syntax

```
Trend.Visible
```

See also

Trend (Page 6621)

Trend.YValueAxis

Description

The "YValueAxis" property references a value axis of the trend.

Type

Object, HmiYValueAxisPart (Page 6583)

Access

Read-write

Syntax

```
Trend.YValueAxis
```

See also

Trend (Page 6621)

YValueAxis (Page 6583)

YValueAxis

Description

YValueAxis (Page 6583)

TrendArea.Visible

Description

The "Visible" property specifies whether the trend area is visible.

Type

Bool

Access

Read-write

Syntax

```
TrendArea.Visible
```

See also

TrendArea (Page 6564)

TrendControl.Visible

Description

The "Visible" property specifies whether the trend view is visible.

Type

Bool

Access

Read-write

Syntax

```
TrendControl.Visible
```

See also

TrendControl (Page 6374)

TrendControl.Width

Description

The "Width" property specifies the width in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
TrendControl.Width
```

See also

TrendControl (Page 6374)

TrendControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the trend view.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be positioned
- CanMaximize (32): Can be maximized
- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

Enable multiple properties by adding the integer values or bit operators.

Access

Read-write

Syntax

```
TrendControl.WindowFlags
```

Example

Adapt the "windowFlags" tag depending on the window configuration:

Copy code

```
var windowFlags = HmiWindowFlag.ShowCaption | HmiWindowFlag.ShowBorder;
if (CanClose){
    windowFlags |= HmiWindowFlag.CanClose;
} else {
    windowFlags &= HmiWindowFlag.CanClose;
}
```

See also

TrendControl (Page 6374)

TrendControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the trend view.

Syntax

```
TrendControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

TrendControl (Page 6374)

TrendControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the trend view.

Syntax

```
TrendControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

TrendControl (Page 6374)

TrendControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
TrendControl.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

`propertyName`

Type: String

Name of the property for which flashing is configured.

`enable`

Type: Bool

Specifies that flashing is activated.

`value`

Optional, type: Variant

Value of the property for which flashing is configured.

`alternateValue`

Optional, type: Variant

Specifies the second value for flashing.

`rate`

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

TrendControl (Page 6374)

TrendControl_OnActivated()

Description

The "OnActivated" event occurs when a trend view receives focus:

- A trend view is selected via the configured tab sequence.
- A trend view that had no focus is clicked/touched.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
TrendControl_OnActivated(item)
```

Context

item

Type: Object

Trend view where the event occurs.

See also

TrendControl (Page 6374)

TrendControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the trend control.

Syntax

```
TrendControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Trend control where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

TrendControl (Page 6374)

TrendControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when a trend view loses focus because the operator presses the <TAB> key or executes another action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

Syntax

```
TrendControl_OnDeactivated(item)
```

Context

item

Type: Object

Trend view where the event occurs.

See also

TrendControl (Page 6374)

TrendControl_OnInitialized()

Description

The "OnInitialized" event occurs when a trend view has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
TrendControl_OnInitialized(item)
```

Context

item

Type: Object

Trend view where the event occurs.

See also

TrendControl (Page 6374)

WebControl

Description

The "WebControl" object represents a web browser in runtime.

Object type

HmiWebControl

Properties

The "WebControl" object has the following properties:

- **BackColor**
  Specifies the background color.
- **Caption**
  Specifies the text that is displayed in the title bar.
- **CaptionColor**
  Specifies the background color of the title bar.
- **CurrentQuality**
  Returns the poorest quality code of all tags which influence the web browser.
- **Enabled**
  Specifies whether the web browser can be operated in runtime.

- **Height**
  Specifies the height.
- **Icon**
  Specifies the icon of the web browser.
- **Layer**
  Returns the screen layer in which the web browser is located.
- **Left**
  Specifies the value of the X coordinate.
- **Margin**
  Specifies the margin.
- **Name**
  Returns the name of the web browser.
- **Parent**
  Returns the higher-level screen object.
- **RenderingTemplate**
  Returns the name of the template from which the web browser was created.
- **ShowFocusVisual**
  Specifies whether the web browser is highlighted when in focus.
- **StatusBar**
  Specifies the information bar of the web browser.
- **StyleItemClass**
  Returns the style which is applied to the web browser.
- **TabIndex**
  Returns the position of the web browser in the tab sequence.
- **ToolBar**
  Specifies the toolbar of the web browser.
- **Top**
  Specifies the value of the Y coordinate.
- **Url**
  Specifies the URL displayed by the web browser.
- **Visible**
  Specifies whether the web browser is visible.
- **Width**
  Specifies the width.
- **WindowFlags**
  Specifies the window configuration of the web browser.

Methods

The "WebControl" object has the following methods:

- **CheckAuthorization()**
  Returns whether the current user is authorized to operate the web browser.
- **FireCommand()**
  Configures the occurrence of an event for an element.
- **PropertyFlashing()**
  Configures flashing of a property.

Events

The "WebControl" object has the following events:

- **OnActivated()**
  Occurs when a web browser receives focus.
- **OnCommandFired()**
  Occurs when the operator has operated an element in the toolbar or information bar of the web browser.
- **OnDeactivated()**
  Occurs when a web browser loses focus.
- **OnInitialized()**
  Occurs when a web browser has been successfully initialized and the data connection to the PLC has been established.

WebControl.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
WebControl.BackColor
```

See also

WebControl (Page 6649)

WebControl.Caption

Description

The "Caption" property specifies the text that is displayed in the title bar.

Type

Object, HmiTextPart

Access

Read-write

Syntax

```
WebControl.Caption
```

See also

WebControl (Page 6649)

Text.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
Text.Font
```

See also

WebControl.Caption (Page 6652)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

Text.Font (Page 6652)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

Text.Font (Page 6652)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

Text.Font (Page 6652)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

Text.Font (Page 6652)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

Text.Font (Page 6652)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

Text.Font (Page 6652)

Text.ForeColor

Description

The "ForeColor" property sets the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
Text.ForeColor
```

See also

WebControl.Caption (Page 6652)

Text.Text

Description

The "Text" property specifies the label.

Type

String

Access

Read-write

Syntax

```
Text.Text
```

See also

WebControl.Caption (Page 6652)

Text.Visible

Description

The "Visible" property specifies whether the text is visible.

Type

Bool

Access

Read-write

Syntax

```
Text.Visible
```

See also

WebControl.Caption (Page 6652)

WebControl.CaptionColor

Description

The "CaptionColor" property specifies the color of the title bar.

Type

UInt32

Access

Read-write

Syntax

```
WebControl.CaptionColor
```

See also

WebControl (Page 6649)

WebControl.CurrentQuality

Description

The "CurrentQuality" property returns the poorest quality code of all tags which influence the web browser.

Type

Int32, HmiQuality

Returns the current quality code:

- None (0): Undefined or not initialized.
- Bad (1): Value cannot be used.
- Uncertain (2): Usable, but the value quality is worse than usual.
- Good (4): Usable, value quality is good.
- UpperLimitViolation (64): Value has exceeded the high limit.
- LowerLimitViolation (128): Value has fallen below the low limit.

Access

Read-only

Syntax

```
WebControl.CurrentQuality
```

See also

WebControl (Page 6649)

WebControl.Enabled

Description

The "Enabled" property specifies whether the web browser can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
WebControl.Enabled
```

See also

WebControl (Page 6649)

WebControl.Height

Description

The "Height" property specifies the height in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
WebControl.Height
```

See also

WebControl (Page 6649)

WebControl.Icon

Description

The "Icon" property specifies the icon of the web browser.

Type

String

Access

Read-write

Syntax

```
WebControl.Icon
```

See also

WebControl (Page 6649)

WebControl.Layer

Description

The "Layer" property returns the screen layer in which the web browser is located.

Type

Object, HmiLayerPart

Access

Read-only

Syntax

```
WebControl.Layer
```

See also

WebControl (Page 6649)

Layer.MaximumZoom

Description

The "MaximumZoom" property specifies the maximum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MaximumZoom
```

See also

WebControl.Layer (Page 6660)

Layer.MinimumZoom

Description

The "MinimumZoom" specifies the minimum zoom up to which the layer can be seen.

Type

Float

Access

Read-write

Syntax

```
Layer.MinimumZoom
```

See also

WebControl.Layer (Page 6660)

Layer.Name

Description

The "Name" property returns the name of the screen layer.

Type

String

Access

Read-only

Syntax

```
Layer.Name
```

See also

WebControl.Layer (Page 6660)

Layer.Visible

Description

The "Visible" property specifies whether the screen layer and contained objects are visible.

Type

Bool

Access

Read-write

Syntax

```
Layer.Visible
```

See also

WebControl.Layer (Page 6660)

WebControl.Left

Description

The "Left" property sets the value of the X coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
WebControl.Left
```

See also

WebControl (Page 6649)

WebControl.Margin

Description

The "Margin" property specifies the margin.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
WebControl.Margin
```

See also

WebControl (Page 6649)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

WebControl.Margin (Page 6663)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

WebControl.Margin (Page 6663)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

WebControl.Margin (Page 6663)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

WebControl.Margin (Page 6663)

WebControl.Name

Description

The "Name" property returns the name of the web browser.

Type

String

Access

Read-only

Syntax

```
WebControl.Name
```

See also

WebControl (Page 6649)

WebControl.Parent

Description

The "Parent" property returns the higher-level screen object (Parent container).

Type

Object, HmiScreenObjectBase (Page 1571)

Access

Read-only

Syntax

```
WebControl.Parent
```

See also

WebControl (Page 6649)

Screen Items (Page 1571)

Screen Items

Description

Screen Items (Page 1571)

WebControl.RenderingTemplate

Description

The "RenderingTemplate" property returns the name of the template from which the web browser was created.

Type

String

Access

Read-only

Syntax

```
WebControl.RenderingTemplate
```

See also

WebControl (Page 6649)

WebControl.ShowFocusVisual

Description

The "ShowFocusVisual" property specifies whether the web browser is highlighted when in focus.

Type

Bool

Access

Read-write

Syntax

```
WebControl.ShowFocusVisual
```

See also

WebControl (Page 6649)

WebControl.StatusBar

Description

The "StatusBar" property specifies the information bar of the web browser.

Type

Object, HmiStatusBarPart

Access

Read-write

Syntax

```
WebControl.StatusBar
```

See also

WebControl (Page 6649)

StatusBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
StatusBar.BackColor
```

See also

WebControl.StatusBar (Page 6668)

StatusBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the information bar.

Type

Object, HmiControlBarElementCollection (Page 6669)

Access

Read-only

Syntax

```
StatusBar.Elements
```

See also

WebControl.StatusBar (Page 6668)

HmiControlBarElementCollection (Page 6669)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

StatusBar.Elements (Page 6669)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6669)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6686)

See also

HmiControlBarElementCollection (Page 6669)

Control Bar Elements (Page 6686)

Control Bar Elements

Description

Control Bar Elements (Page 6686)

StatusBar.Enabled

Description

The "Enabled" property specifies whether the information bar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Enabled
```

See also

WebControl.StatusBar (Page 6668)

StatusBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
StatusBar.Font
```

See also

WebControl.StatusBar (Page 6668)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

StatusBar.Font (Page 6672)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

StatusBar.Font (Page 6672)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

StatusBar.Font (Page 6672)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

StatusBar.Font (Page 6672)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

StatusBar.Font (Page 6672)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

StatusBar.Font (Page 6672)

StatusBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the information bar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
StatusBar.Margin
```

See also

WebControl.StatusBar (Page 6668)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

StatusBar.Margin (Page 6676)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

StatusBar.Margin (Page 6676)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

StatusBar.Margin (Page 6676)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

StatusBar.Margin (Page 6676)

StatusBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the information bar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
StatusBar.Padding
```

See also

WebControl.StatusBar (Page 6668)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

StatusBar.Padding (Page 6678)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

StatusBar.Padding (Page 6678)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

StatusBar.Padding (Page 6678)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

StatusBar.Padding (Page 6678)

StatusBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.ShowToolTips
```

See also

WebControl.StatusBar (Page 6668)

StatusBar.Visible

Description

The "Visible" property specifies whether the information bar is visible.

Type

Bool

Access

Read-write

Syntax

```
StatusBar.Visible
```

See also

WebControl.StatusBar (Page 6668)

WebControl.StyleItemClass

Description

The "StyleItemClass" property returns the style which is applied to the web browser.

Type

String

Access

Read-only

Syntax

```
WebControl.StyleItemClass
```

See also

WebControl (Page 6649)

WebControl.TabIndex

Description

The "TabIndex" property returns the position of the web browser in the tab sequence.

Type

UInt16

Access

Read-only

Syntax

```
WebControl.TabIndex
```

See also

WebControl (Page 6649)

WebControl.ToolBar

Description

The "ToolBar" property specifies the toolbar of the web browser.

Type

Object, HmiToolBarPart

Access

Read-write

Syntax

```
WebControl.ToolBar
```

See also

WebControl (Page 6649)

ToolBar.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ToolBar.BackColor
```

See also

WebControl.ToolBar (Page 6683)

ToolBar.Elements

Description

The "Elements" property returns a list of the basic elements (e.g., buttons or labels) of the toolbar.

Type

Object, HmiControlBarElementCollection (Page 6684)

Access

Read-only

Syntax

```
ToolBar.Elements
```

See also

WebControl.ToolBar (Page 6683)

HmiControlBarElementCollection (Page 6684)

HmiControlBarElementCollection

Description

The "HmiControlBarElementCollection" object is a list of all the basic elements (e.g. buttons or labels) of an information bar or toolbar.

You reference a "HmiControlBarElementCollection" object via the property `StatusBar.Elements` or `ToolBar.Elements`

Use

The "HmiControlBarElementCollection" object is a list which can be counted and enumerated. You can access the "HmiControlBarElementCollection" list using the index or the tag names.

Object type

HmiControlBarElementCollection

Properties

The "HmiControlBarElementCollection" object has the following properties:

- **Count**
  Returns the number of basic elements of the "HmiControlBarElementCollection" list.

Methods

The "HmiControlBarElementCollection" object has the following methods:

- **Item()**
  Returns a basic element of the "HmiControlBarElementCollection" list.

See also

ToolBar.Elements (Page 6684)

HmiControlBarElementCollection.Count

Description

The "Count" property returns the number of basic elements (e.g., buttons or labels) of the "HmiControlBarElementCollection" list.

Type

UInt32

Access

Read-only

Syntax

```
HmiControlBarElementCollection.Count
```

See also

HmiControlBarElementCollection (Page 6684)

HmiControlBarElementCollection.Item()

Description

The "Item" method returns a basic element (e.g., button or label) of the "HmiControlBarElementCollection" list.

Syntax

```
HmiControlBarElementCollection[.Item](HmiControlBarElementName)
```

Note

The `.Item` part of the expression is not required. The "Item" method is the standard method of the "HmiControlBarElementCollection" object.

Parameters

`HmiControlBarElementName`

Type: String

Name of the element

Return value

Object, HmiControlBarElementPartBase (Page 6686)

See also

HmiControlBarElementCollection (Page 6684)

Control Bar Elements (Page 6686)

Control Bar Elements

ControlBarButton

Description

The "ControlBarButton" object represents a button of an information bar or toolbar.

You can reference the "ControlBarButton" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarButtonPart

Properties

The "ControlBarButton" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the label of the button.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the button can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the button.
- **Mapping**
  Returns the function of the button.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.

- **Operability**
  Returns whether the button is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the button is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the button is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarButton.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBackColor
```

See also

ControlBarButton (Page 6686)

ControlBarButton.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.AlternateBorderColor
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the button of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarButton.Authorization
```

See also

ControlBarButton (Page 6686)

ControlBarButton.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BackColor
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Badge

Description

The "Badge" property specifies the label of the button.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarButton.Badge
```

See also

ControlBarButton (Page 6686)

ControlBarButton.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.BorderColor
```

See also

ControlBarButton (Page 6686)

ControlBarButton.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarButton.BorderWidth
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarButton.Content
```

See also

ControlBarButton (Page 6686)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarButton.Content (Page 6692)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarButton.Content (Page 6692)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarButton.Content (Page 6692)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarButton.Content (Page 6692)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarButton.Content (Page 6692)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarButton.Content (Page 6692)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarButton.Content (Page 6692)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarButton.Content (Page 6692)

ControlBarButton.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarButton.CustomID
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Enabled

Description

The "Enabled" property specifies whether the button can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Enabled
```

See also

ControlBarButton (Page 6686)

ControlBarButton.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.ForeColor
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Graphic

Description

The "Graphic" property specifies the graphic of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Graphic
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Height

Description

The "Height" property specifies the height of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Height
```

See also

ControlBarButton (Page 6686)

ControlBarButton.HotKey

Description

The "HotKey" property returns the hotkey specified for the button. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
ControlBarButton.HotKey
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent

- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms

- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarButton.Mapping
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Margin

Description

The "Margin" property specifies the surrounded outer distance of the button.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarButton.Margin
```

See also

ControlBarButton (Page 6686)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarButton.Margin (Page 6702)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarButton.Margin (Page 6702)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarButton.Margin (Page 6702)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarButton.Margin (Page 6702)

ControlBarButton.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumHeight
```

See also

ControlBarButton (Page 6686)

ControlBarButton.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MaximumWidth
```

See also

ControlBarButton (Page 6686)

ControlBarButton.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumHeight
```

See also

ControlBarButton (Page 6686)

ControlBarButton.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.MinimumWidth
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Operability

Description

The "Operability" property returns whether the button is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarButton.Operability
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Padding

Description

The "Padding" property specifies the distance of the content from the border of the button.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarButton.Padding
```

See also

ControlBarButton (Page 6686)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarButton.Padding (Page 6707)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarButton.Padding (Page 6707)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarButton.Padding (Page 6707)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarButton.Padding (Page 6707)

ControlBarButton.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the button can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarButton.RequireExplicitUnlock
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Text

Description

The "Text" property specifies the label of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.Text
```

See also

ControlBarButton (Page 6686)

ControlBarButton.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the button.

Type

String

Access

Read-write

Syntax

```
ControlBarButton.ToolTipText
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Visible

Description

The "Visible" property specifies whether the button is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarButton.Visible
```

See also

ControlBarButton (Page 6686)

ControlBarButton.Width

Description

The "Width" property specifies the width of the button in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarButton.Width
```

See also

ControlBarButton (Page 6686)

ControlBarDisplay

Description

The "ControlBarDisplay" object represents a display area for text and graphics of an information bar or toolbar.

You can reference the "ControlBarDisplay" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarDisplayPart

Properties

The "ControlBarDisplay" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **Content**
  Specifies display options for text and graphics.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the display area is enabled in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the display area.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the display area is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the display area is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.

- **Visible**
  Specifies whether the display area is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarDisplay.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the display.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarDislay.Authorization
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarDislay.Content
```

See also

ControlBarDisplay (Page 6712)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarDisplay.Content (Page 6714)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.
- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarDisplay.Content (Page 6714)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarDisplay.Content (Page 6714)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarDisplay.Content (Page 6714)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarDisplay.Content (Page 6714)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarDisplay.Content (Page 6714)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarDisplay.Content (Page 6714)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarDisplay.Content (Page 6714)

ControlBarDisplay.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarDislay.CustomID
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Enabled

Description

The "Enabled" property specifies whether the display is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Enabled
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.ForeColor
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Graphic

Description

The "Graphic" property specifies the graphic of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Graphic
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Height

Description

The "Height" property specifies the height of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Height
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display

- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms

- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarDislay.Mapping
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the display.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarDislay.Margin
```

See also

ControlBarDisplay (Page 6712)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarDisplay.Margin (Page 6724)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarDisplay.Margin (Page 6724)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarDisplay.Margin (Page 6724)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarDisplay.Margin (Page 6724)

ControlBarDisplay.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumHeight
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MaximumWidth
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumHeight
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.MinimumWidth
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Operability

Description

The "Operability" property returns whether the display is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarDislay.Operability
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the display.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarDislay.Padding
```

See also

ControlBarDisplay (Page 6712)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarDisplay.Padding (Page 6729)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarDisplay.Padding (Page 6729)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarDisplay.Padding (Page 6729)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarDisplay.Padding (Page 6729)

ControlBarDisplay.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the display can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarDislay.RequireExplicitUnlock
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Text

Description

The "Text" property specifies the label of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.Text
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the display.

Type

String

Access

Read-write

Syntax

```
ControlBarDislay.ToolTipText
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Visible

Description

The "Visible" property specifies whether the display is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarDislay.Visible
```

See also

ControlBarDisplay (Page 6712)

ControlBarDisplay.Width

Description

The "Width" property specifies the width of the display in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarDislay.Width
```

See also

ControlBarDisplay (Page 6712)

ControlBarLabel

Description

The "ControlBarLabel" object represents an identifier of an information bar or toolbar.

You can reference the "ControlBarLabel" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarLabelPart

Properties

The "ControlBarLabel" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the identifier can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the identifier.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the identifier is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the identifier is only operable while the corresponding button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.

- **Visible**
  Specifies whether the identifier is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarLabel.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the identifier.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarLabel.Authorization
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarLabel.CustomID
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Enabled

Description

The "Enabled" property specifies whether the identifier is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Enabled
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.ForeColor
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Height

Description

The "Height" property specifies the height of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Height
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.HorizontalTextAlignment
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms

- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export

- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarLabel.Mapping
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the identifier.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarLabel.Margin
```

See also

ControlBarLabel (Page 6734)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarLabel.Margin (Page 6741)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarLabel.Margin (Page 6741)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarLabel.Margin (Page 6741)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarLabel.Margin (Page 6741)

ControlBarLabel.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumHeight
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MaximumWidth
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumHeight
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.MinimumWidth
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Operability

Description

The "Operability" property returns whether the identifier is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarLabel.Operability
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Padding

Description

The "Padding" property specifies the spacing of the content from the frame of the identifier.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarLabel.Padding
```

See also

ControlBarLabel (Page 6734)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarLabel.Padding (Page 6746)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarLabel.Padding (Page 6746)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarLabel.Padding (Page 6746)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarLabel.Padding (Page 6746)

ControlBarLabel.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the identifier can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarLabel.RequireExplicitUnlock
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Text

Description

The "Text" property specifies the label of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.Text
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the identifier.

Type

String

Access

Read-write

Syntax

```
ControlBarLabel.ToolTipText
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarLabel.VerticalTextAlignment
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Visible

Description

The "Visible" property specifies whether the identifier is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarLabel.Visible
```

See also

ControlBarLabel (Page 6734)

ControlBarLabel.Width

Description

The "Width" property specifies the width of the identifier in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarLabel.Width
```

See also

ControlBarLabel (Page 6734)

ControlBarSeparator

Description

The "ControlBarSeparator" object represents a separator of an information bar or toolbar.

You can reference the "ControlBarSeparator" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarSeparatorPart

Properties

The "ControlBarSeparator" object has the following properties:

- **Authorization**
  Returns the operator authorization.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the disconnector can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **Mapping**
  Returns the function of the separator.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.

- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the separator is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the disconnector is only operable while the associated button is being pressed.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the disconnector is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarSeparator.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the separator.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarSeparator.Authorization
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarSeparator.CustomID
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Enabled

Description

The "Enabled" property specifies whether the separator is active in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Enabled
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.ForeColor
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Height

Description

The "Height" property specifies the height of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Height
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup

- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarSeparator.Mapping
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the separator.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarSeparator.Margin
```

See also

ControlBarSeparator (Page 6752)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarSeparator.Margin (Page 6758)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarSeparator.Margin (Page 6758)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarSeparator.Margin (Page 6758)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarSeparator.Margin (Page 6758)

ControlBarSeparator.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumHeight
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MaximumWidth
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumHeight
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.MinimumWidth
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Operability

Description

The "Operability" property returns whether the separator is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarSeparator.Operability
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the separator.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarSeparator.Padding
```

See also

ControlBarSeparator (Page 6752)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarSeparator.Padding (Page 6763)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarSeparator.Padding (Page 6763)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarSeparator.Padding (Page 6763)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarSeparator.Padding (Page 6763)

ControlBarSeparator.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the disconnector can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarSeparator.RequireExplicitUnlock
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the separator.

Type

String

Access

Read-write

Syntax

```
ControlBarSeparator.ToolTipText
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Visible

Description

The "Visible" property specifies whether the separator is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarSeparator.Visible
```

See also

ControlBarSeparator (Page 6752)

ControlBarSeparator.Width

Description

The "Width" property specifies the width of the separator in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarSeparator.Width
```

See also

ControlBarSeparator (Page 6752)

ControlBarTextBox

Description

The "ControlBarTextBox" object represents a text box of an information bar or toolbar.

You can reference the "ControlBarTextBox" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarTextBoxPart

Properties

The "ControlBarTextBox" object has the following properties:

- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the text box can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Height**
  Specifies the height.
- **HorizontalTextAlignment**
  Specifies the horizontal alignment of the text.
- **Mapping**
  Returns the function of the text box.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.

- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the text box is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **Readonly**
  Specifies whether the text box is write-protected.
- **RequireExplicitUnlock**
  Returns whether the text box is only operable while the associated button is being pressed.
- **Text**
  Specifies the labeling.
- **ToolTipText**
  Specifies the tooltip text.
- **VerticalTextAlignment**
  Specifies the vertical alignment of the text.
- **Visible**
  Specifies whether the text box is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarTextBox.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.AlternateBorderColor
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the text box of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarTextBox.Authorization
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BackColor
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.BorderColor
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarTextBox.BorderWidth
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarTextBox.CustomID
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Enabled

Description

The "Enabled" property specifies whether the text box can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Enabled
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.ForeColor
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Height

Description

The "Height" property specifies the height of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Height
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.HorizontalTextAlignment
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment
- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms

- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms
- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarTextBox.Mapping
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the text box.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarTextBox.Margin
```

See also

ControlBarTextBox (Page 6768)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarTextBox.Margin (Page 6777)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarTextBox.Margin (Page 6777)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarTextBox.Margin (Page 6777)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarTextBox.Margin (Page 6777)

ControlBarTextBox.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumHeight
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MaximumWidth
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumHeight
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.MinimumWidth
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Operability

Description

The "Operability" property returns whether the text box is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarTextBox.Operability
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the text box.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarTextBox.Padding
```

See also

ControlBarTextBox (Page 6768)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarTextBox.Padding (Page 6782)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarTextBox.Padding (Page 6782)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarTextBox.Padding (Page 6782)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarTextBox.Padding (Page 6782)

ControlBarTextBox.ReadOnly

Description

The "ReadOnly" property specifies whether the text box is read-only.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.ReadOnly
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the text box can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarTextBox.RequireExplicitUnlock
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Text

Description

The "Text" property specifies the label of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.Text
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the text box.

Type

String

Access

Read-write

Syntax

```
ControlBarTextBox.ToolTipText
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
ControlBarTextBox.VerticalTextAlignment
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Visible

Description

The "Visible" property specifies whether the text box is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarTextBox.Visible
```

See also

ControlBarTextBox (Page 6768)

ControlBarTextBox.Width

Description

The "Width" property specifies the width of the text box in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarTextBox.Width
```

See also

ControlBarTextBox (Page 6768)

ControlBarToggleSwitch

Description

The "ControlBarToggleSwitch" object represents a switch of an information bar or toolbar.

You can reference the "ControlBarToggleSwitch" object via the "HmiControlBarElementCollection" list.

Object type

HmiControlBarToggleSwitchPart

Properties

The "ControlBarToggleSwitch" object has the following properties:

- **AlternateBackColor**
  Specifies the second color for a color gradient.
- **AlternateBorderColor**
  Specifies the second border color which is displayed for line styles such as "Dash".
- **AlternateGraphic**
  Specifies the graphic for the "pressed" state.
- **AlternateText**
  Specifies the text for the "pressed" state.
- **Authorization**
  Returns the operator authorization.
- **BackColor**
  Specifies the background color.
- **Badge**
  Specifies the identifier of the switch.
- **BorderColor**
  Specifies the line color.
- **BorderWidth**
  Specifies the line thickness.
- **Content**
  Specifies display options for text and graphics.

- **CustomID**
  Returns the custom ID.
- **Enabled**
  Specifies whether the switch can be operated in runtime.
- **ForeColor**
  Specifies the font color.
- **Graphic**
  Specifies the graphic.
- **Height**
  Specifies the height.
- **HotKey**
  Returns the hotkey specified for the switch.
- **IsAlternateState**
  Specifies the current state of the switch.
- **Mapping**
  Returns the function of the switch.
- **Margin**
  Specifies the margin.
- **MaximumHeight**
  Specifies the maximum height.
- **MaximumWidth**
  Specifies the maximum width.
- **MinimumHeight**
  Specifies the minimum height.
- **MinimumWidth**
  Specifies the minimum width.
- **Operability**
  Returns whether the switch is operable.
- **Padding**
  Specifies the distance of the content from the border.
- **RequireExplicitUnlock**
  Returns whether the switch is only operable while the associated button is being pressed.
- **Text**
  Specifies the label.
- **ToolTipText**
  Specifies the tooltip text.
- **Visible**
  Specifies whether the switch is visible.
- **Width**
  Specifies the width.

Methods

--

ControlBarToggleSwitch.AlternateBackColor

Description

The "AlternateBackColor" property specifies the second background color for a color gradient.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBackColor
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.AlternateBorderColor

Description

The "AlternateBorderColor" property specifies the second border color used with line styles such as "Dash".

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateBorderColor
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.AlternateGraphic

Description

The "AlternateGraphic" property specifies the graphic of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateGraphic
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.AlternateText

Description

The "AlternateText" property specifies the text of the switch for the "On" state.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.AlternateText
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Authorization

Description

The "Authorization" property returns the operator authorization (HmiFunctionRight) that the logged-in operator must have to use the switch of an information bar or toolbar.

Type

Object, HmiFunctionRight

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Authorization
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.BackColor

Description

The "BackColor" property specifies the background color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BackColor
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Badge

Description

The "Badge" property specifies the identifier of the switch.

Type

Object, HmiBadgePart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Badge
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.BorderColor

Description

The "BorderColor" property specifies the border color.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderColor
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.BorderWidth

Description

The "BorderWidth" property specifies the border thickness.

Type

UInt8

Access

Read-write

Syntax

```
ControlBarToggleSwitch.BorderWidth
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Content

Description

The "Content" property specifies display options for text and graphics.

Type

Object, HmiContentPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Content
```

See also

ControlBarToggleSwitch (Page 6788)

Content.ContentMode

Description

The "ContentMode" property specifies whether text, graphics, or both are used for the display.

Type

Int32, HmiContentMode

Specifies the appearance of text and graphics:

- GraphicOrText (0): Graphic has priority. If no graphic is available, text is used.
- GraphicAndText (1): Text and graphics
- Text (2): Text only
- Graphic (3): Graphic only

Access

Read-write

Syntax

```
Content.ContentMode
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.GraphicStretchMode

Description

The "GraphicStretchMode" property defines the type of scaling of the graphic.

Type

Int32, HmiGraphicStretchMode

Specifies the scaling of the graphic:

- None (0): The graphic is shown in the original size and centered.
- Fill (1): Graphic is displayed in the available space. Aspect ratio is adjusted, but not scaled.

- Uniform (2): Graphic is displayed in the available space. Aspect ratio is not changed.
- UniformToFill (3): Graphic is displayed in the available space. The rest is truncated. Aspect ratio is not changed.
- Tiled (4): The graphic is shown in the original size and repeated in tiles.

Access

Read-write

Syntax

```
Content.GraphicStretchMode
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.HorizontalTextAlignment

Description

The "HorizontalTextAlignment" property specifies the horizontal alignment of the text.

Type

Int32, HmiHorizontalAlignment

Specifies the horizontal alignment:

- Left (0): Left
- Center (1): Centered
- Right (2): Right
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.HorizontalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.Spacing

Description

The "Spacing" property specifies the spacing between graphics and text.

Type

UInt32

Access

Read-write

Syntax

```
Content.Spacing
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.SplitRatio

Description

The "SplitRatio" property specifies how much space the graphic takes up in relation to the text. A split ratio of 0.7 means that the graphic takes up 70% of the space.

Type

Float

Access

Read-write

Syntax

```
Content.SplitRatio
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.TextPosition

Description

The "TextPosition" property specifies the position of the text in comparison to the graphic.

Type

Int32, HmiTextPosition

Specifies the position of the text:

- Left (0): Left
- Right (1): Right
- Top (2): Above
- Bottom (3): Below
- Behind (4): Behind
- InFront (5): In front

Access

Read-write

Syntax

```
Content.TextPosition
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.TextTrimming

Description

The "TextTrimming" property specifies the type of text trimming if space is not sufficient.

Type

Int32, HmiTextTrimming

Specifies the text trimming:

- None (0): No trimming
- CharacterEllipsis (1): Trimming of the text end with character ellipsis ("...")

Access

Read-write

Syntax

```
Content.TextTrimming
```

See also

ControlBarToggleSwitch.Content (Page 6794)

Content.VerticalTextAlignment

Description

The "VerticalTextAlignment" property specifies the vertical alignment of the text.

Type

Int32, HmiVerticalAlignment

Specifies the vertical alignment:

- Top (0): Top
- Center (1): Centered
- Bottom (2): Bottom
- Stretch (3): Stretched

Access

Read-write

Syntax

```
Content.VerticalTextAlignment
```

See also

ControlBarToggleSwitch.Content (Page 6794)

ControlBarToggleSwitch.CustomID

Description

The "CustomID" property returns the custom ID for identification of an element of an information bar or toolbar.

Type

Int32

Access

Read-only

Syntax

```
ControlBarToggleSwitch.CustomID
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Enabled

Description

The "Enabled" property specifies whether the switch can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Enabled
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.ForeColor

Description

The "ForeColor" property specifies the font color of the text.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ForeColor
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Graphic

Description

The "Graphic" property specifies the graphic of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Graphic
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Height

Description

The "Height" property specifies the height of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Height
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.HotKey

Description

The "HotKey" property returns the hotkey specified for the switch. Hotkeys are unique within a screen. The hotkeys are activated from the active (local) screen window down to the top level screen window.

Type

UInt16

Access

Read-only

Syntax

```
Button.HotKey
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.IsAlternateState

Description

The "IsAlternateState" property specifies the current state of the switch.

- True: Switch is in "On" state.
- False: Switch is in "Off" state.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.IsAlternateState
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Mapping

Description

The "Mapping" property returns the function of an element of an information bar or toolbar.

Type

Int32, HmiAlarmControlID

Function of an element of the alarm control:

- None (0): None
- Help (1): Show help
- Configuration (2): Configuration dialog
- AlarmView (3): Alarm control
- AlarmStatisticsView (5): Alarm statistics display
- AlarmAnnunciator (7): Alarm annunciator
- SingleAcknowledgment (8): Single-mode acknowledgment
- GroupAcknowledgment (9): Group acknowledgment

- AlwaysShowRecent (10): Show recent
- SelectionDisplay (11): Selection display
- DisplayOptionsSetup (12): Display options setup
- DisabledAlarmsSetup (13): Configuration of the locked alarms
- FirstLine (14): First row
- PreviousLine (15): Previous line
- NextLine (16): Next line
- LastLine (17): Last line
- InfoTextSetup (18): Info text setup
- CommentsSetup (19): Comments setup
- LoopInAlarm (20): Loop-in-alarm
- DisableAlarm (21): Lock alarm
- EnableAlarm (22): Release alarm
- ShelveAlarm (23): Shelve alarm
- UnshelveAlarm (24): Unshelve alarm
- SortSetup (25): Sorting setup
- TimeBaseSetup (26): Time base setup
- CopyLines (27): Copy lines
- PreviousPage (28): Previous page
- NextPage (29): Next page
- TimeBase (256): Timebase
- Date (257): Date
- Time (258): Time
- PendingAlarms (259): Pending alarms
- Alarms (260): Alarms
- PendingAcknowledgeableAlarms (261): Pending acknowledgeable alarms
- PendingShelvedAlarms (262): Pending shelved alarms
- Selection (263): Selection
- DisplayOption (264): Display options
- Disabled (265): Disabled
- HasPendingShelvedAlarms (272): Pending shelved alarms available
- ConnectionStatus (273): Connection status
- Print (30): Print
- ShowActiveAlarms (31): Show active alarms
- ShowLoggedAlarms (32): Show logged alarms

- ShowLoggedAlarmsUpdated (33): Show and update logged alarms
- ShowDefinedAlarms (34): Show defined alarms
- SingleReset (35): Single confirm
- Export (36): Export
- MoveToNextAcknowledgeableAlarm (37): Skip to the next alarm that requires acknowledgment
- StartTime (274): Start time
- EndTime (275): End time
- CurrentContextHint (276): Note on current context
- SelectContext (38): Select context
- StatusText (277): Status text
- Custom (65536): Reserved for user-defined functions. The "CustomID" property can be used for identification.
- PendingResettableAlarms (278): Pending engaged alarms
- StatisticsSetup (39): Show alarm statistics settings
- MaximumRecordsExceeded (279): Number of logged alarms exceeds the value of AlarmStatisticsSettings.MaximumRecords.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Mapping
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the switch.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Margin
```

See also

ControlBarToggleSwitch (Page 6788)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ControlBarToggleSwitch.Margin (Page 6805)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ControlBarToggleSwitch.Margin (Page 6805)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ControlBarToggleSwitch.Margin (Page 6805)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ControlBarToggleSwitch.Margin (Page 6805)

ControlBarToggleSwitch.MaximumHeight

Description

The "MaximumHeight" property specifies the maximum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumHeight
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.MaximumWidth

Description

The "MaximumWidth" property specifies the maximum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MaximumWidth
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.MinimumHeight

Description

The "MinimumHeight" property specifies the minimum height.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumHeight
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.MinimumWidth

Description

The "MinimumWidth" property specifies the minimum width.

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.MinimumWidth
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Operability

Description

The property "Operability" returns whether the switch is operable.

Type

Int32, HmiOperability

Returns the operability:

- Operable (0): Operable
- DisabledProgrammatically (1): Not operable because disabled.
- NoAuthorization (2): Not operable because operator authorization is missing.
- NoExplicitUnlock (4): Not operable because unlock button is not pressed.

Access

Read-only

Syntax

```
ControlBarToggleSwitch.Operability
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the switch.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Padding
```

See also

ControlBarToggleSwitch (Page 6788)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ControlBarToggleSwitch.Padding (Page 6810)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ControlBarToggleSwitch.Padding (Page 6810)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ControlBarToggleSwitch.Padding (Page 6810)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ControlBarToggleSwitch.Padding (Page 6810)

ControlBarToggleSwitch.RequireExplicitUnlock

Description

The "RequireExplicitUnlock" property returns whether the switch can only be operated while the associated button is being pressed.

Type

Bool

Access

Read-only

Syntax

```
ControlBarToggleSwitch.RequireExplicitUnlock
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Text

Description

The "Text" property specifies the label of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Text
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.ToolTipText

Description

The "ToolTipText" property specifies the text of the tooltip of the switch.

Type

String

Access

Read-write

Syntax

```
ControlBarToggleSwitch.ToolTipText
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Visible

Description

The "Visible" property specifies whether the switch is visible.

Type

Bool

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Visible
```

See also

ControlBarToggleSwitch (Page 6788)

ControlBarToggleSwitch.Width

Description

The "Width" property specifies the width of the switch in DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
ControlBarToggleSwitch.Width
```

See also

ControlBarToggleSwitch (Page 6788)

ToolBar.Enabled

Description

The "Enabled" property specifies whether the toolbar can be operated in runtime.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Enabled
```

See also

WebControl.ToolBar (Page 6683)

ToolBar.Font

Description

The "Font" property specifies the font of the text.

Type

Object, HmiFontPart

Access

Read-write

Syntax

```
ToolBar.Font
```

See also

WebControl.ToolBar (Page 6683)

Font.Italic

Description

The "Italic" property specifies whether the text is displayed in italics.

Type

Bool

Access

Read-write

Syntax

```
Font.Italic
```

See also

ToolBar.Font (Page 6816)

Font.Name

Description

The "Name" property specifies the name of the character set used.

Type

String

Access

Read-write

Syntax

```
Font.Name
```

See also

ToolBar.Font (Page 6816)

Font.Size

Description

The "Size" property specifies the font size in DIU (Device Independent Unit).

Type

Float

Access

Read-write

Syntax

```
Font.Size
```

See also

ToolBar.Font (Page 6816)

Font.StrikeOut

Description

The "StrikeOut" property specifies whether font is struck through.

Type

Int32, HmiFontStrikeOut

Specifies the type of strikethrough font:

- None (0): None
- Single (1): Single

Access

Read-write

Syntax

```
Font.StrikeOut
```

See also

ToolBar.Font (Page 6816)

Font.Underline

Description

The "Underline" property specifies whether the font is underlined.

Type

Bool

Access

Read-write

Syntax

```
Font.Underline
```

See also

ToolBar.Font (Page 6816)

Font.Weight

Description

The "Weight" property specifies the font thickness.

Type

Int32, HmiFontWeight

Specifies the font weight:

- None (0): None
- Light (300): Light
- Normal (400): Normal
- SemiBold (600): Semi-bold
- Bold (700): Bold

Access

Read-write

Syntax

```
Font.Weight
```

See also

ToolBar.Font (Page 6816)

ToolBar.Margin

Description

The "Margin" property specifies the surrounded outer spacing of the toolbar.

Type

Object, HmiMarginPart

Access

Read-write

Syntax

```
ToolBar.Margin
```

See also

WebControl.ToolBar (Page 6683)

Margin.Bottom

Description

The "Bottom" property specifies the spacing to the bottom.

Type

Int32

Access

Read-write

Syntax

```
Margin.Bottom
```

See also

ToolBar.Margin (Page 6820)

Margin.Left

Description

The "Left" property specifies the spacing to the left.

Type

Int32

Access

Read-write

Syntax

```
Margin.Left
```

See also

ToolBar.Margin (Page 6820)

Margin.Right

Description

The "Right" property specifies the spacing to the right.

Type

Int32

Access

Read-write

Syntax

```
Margin.Right
```

See also

ToolBar.Margin (Page 6820)

Margin.Top

Description

The "Top" property specifies the spacing to the top.

Type

Int32

Access

Read-write

Syntax

```
Margin.Top
```

See also

ToolBar.Margin (Page 6820)

ToolBar.Padding

Description

The "Padding" property specifies the spacing of the content from the border of the toolbar.

Type

Object, HmiPaddingPart

Access

Read-write

Syntax

```
ToolBar.Padding
```

See also

WebControl.ToolBar (Page 6683)

Padding.Bottom

Description

The "Bottom" property specifies the bottom distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Bottom
```

See also

ToolBar.Padding (Page 6822)

Padding.Left

Description

The "Left" property specifies the left distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Left
```

See also

ToolBar.Padding (Page 6822)

Padding.Right

Description

The "Right" property specifies the right distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Right
```

See also

ToolBar.Padding (Page 6822)

Padding.Top

Description

The "Top" property specifies the top distance of the content from the border.

Type

Int32

Access

Read-write

Syntax

```
Padding.Top
```

See also

ToolBar.Padding (Page 6822)

ToolBar.ShowToolTips

Description

The "ShowToolTips" property specifies whether tooltips are displayed.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.ShowToolTips
```

See also

WebControl.ToolBar (Page 6683)

ToolBar.UseHotKeys

Description

The "UseHotKeys" property specifies whether the shortcut keys for the buttons in the toolbar are activated.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.UseHotKeys
```

See also

WebControl.ToolBar (Page 6683)

ToolBar.Visible

Description

The "Visible" property specifies whether the toolbar is visible.

Type

Bool

Access

Read-write

Syntax

```
ToolBar.Visible
```

See also

WebControl.ToolBar (Page 6683)

WebControl.Top

Description

The "Top" property sets the value of the Y coordinate in the DIU (Device Independent Unit).

Type

Int32

Access

Read-write

Syntax

```
WebControl.Top
```

See also

WebControl (Page 6649)

WebControl.Url

Description

The "Url" property specifies the URL displayed by the web browser.

Type

String

Access

Read-write

Syntax

```
WebControl.Url
```

See also

WebControl (Page 6649)

WebControl.Visible

Description

The "Visible" property specifies whether the web browser is visible.

Type

Bool

Access

Read-write

Syntax

```
WebControl.Visible
```

See also

WebControl (Page 6649)

WebControl.Width

Description

The "Width" property specifies the width in the DIU (Device Independent Unit).

Type

UInt32

Access

Read-write

Syntax

```
WebControl.Width
```

See also

WebControl (Page 6649)

WebControl.WindowFlags

Description

The "WindowFlags" property specifies the window configuration of the web browser.

Type

Int32, HmiWindowFlag

Specifies the window configuration:

- None (0): Use default setting of the object
- ShowCaption (1): Show title
- ShowBorder (2): Show border
- AlwaysOnTop (4): Always on top
- CanSize (8): Can be sized
- CanMove (16): Can be moved
- CanMaximize (32): Can be maximized

- CanClose (64): Can be closed
- AlwaysInParent (128): Position always inside the surrounding object

Note

You can enable multiple properties by adding integer values or bit operators.

Access

Read-write

Syntax

```
WebControl.WindowFlags
```

See also

WebControl (Page 6649)

WebControl.CheckAuthorization()

Description

The "CheckAuthorization" method returns whether the current user is authorized to operate the web browser.

Syntax

```
WebControl.CheckAuthorization()
```

Parameters

--

Return value

Bool

Example

Set the background of all screen objects that start with "btn" and cannot be operated to light gray:

Copy code

```
for (const screenItem of Screen.Items) {
    if (screenItem.Name.startsWith('btn') && !screenItem.CheckAuthorization()) {
        screenItem.BackColor = 0xFFAAAAAA; // light grey
    }
}
```

See also

WebControl (Page 6649)

WebControl.FireCommand()

Description

The "FireCommand" method executes the command of an element in the toolbar or information bar (control bar element, e.g. a button) of the browser.

Syntax

```
WebControl.FireCommand(commandId, custom)
```

Parameters

commandId

Type: Int32

ID of the element of the toolbar or information bar

custom

Type: Bool

Type of element ID:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

Return value

--

See also

WebControl (Page 6649)

WebControl.PropertyFlashing()

Description

The "PropertyFlashing" method configures the flashing of a property. Flashing refers to the change between two values of a property.

Syntax

```
WebControl.PropertyFlashing(propertyName, enable[, value][,
alternateValue][, rate])
```

Parameters

propertyName

Type: String

Name of the property for which flashing is configured.

enable

Type: Bool

Specifies that flashing is activated.

value

Optional, type: Variant

Value of the property for which flashing is configured.

alternateValue

Optional, type: Variant

Specifies the second value for flashing.

rate

Optional, type: Int32, HmiFlashingRate

Specifies the flash rate:

- Slow (0): Slow, 2 s
- Medium (1): Medium, 1 s
- Fast (2): Fast, 500 ms

Note

If the parameter is missing, "Medium" is used.

Return value

Bool

See also

WebControl (Page 6649)

WebControl_OnActivated()

Description

The "OnActivated" event occurs when a web browser receives focus:

- A web browser is selected via the configured tab sequence.
- A web browser that had no focus is clicked/tapped.

The "OnActivated" event is only used to detect whether an object has been selected and is receiving focus. The event is not triggered if the object already had the focus.

The event does not trigger a password prompt. For this reason, do not use the "OnActivated" event if you want to configure access protection on the function call of the object.

Syntax

```
Ellipse_OnActivated(item)
```

Context

item

Type: Object

Web browser where the event occurs.

See also

WebControl (Page 6649)

WebControl_OnCommandFired()

Description

The "OnCommandFired" event occurs when the operator has operated an element in the toolbar or information bar (control bar element, e.g. a button) of the web browser.

Syntax

```
WebControl_OnCommandFired(item, commandId, custom)
```

Context

item

Type: Object

Web browser where the event occurs.

commandId

Type: DInt

ID of the element of the toolbar or information bar that was operated.

custom

Type: Bool

Specifies the type of the ID of the operated element:

- True: Custom ID
- False: ID of a standard function of an element (ControlID)

See also

WebControl (Page 6649)

WebControl_OnDeactivated()

Description

The "OnDeactivated" event occurs when a web browser loses focus because the operator presses the <TAB> key or executes a different action with the mouse.

The "OnDeactivated" event is only used to detect whether an object was deselected.

The event does not trigger a password prompt. For this reason, do not use the "OnDeactivated" event if you want to configure access protection on the function call of the object.

System functions or user-defined functions on the "OnDeactivated" event are not executed when a screen is closed.

Syntax

```
WebControl_OnDeactivated(item)
```

Context

item

Type: Object

Web browser where the event occurs.

See also

WebControl (Page 6649)

WebControl_OnInitialized()

Description

The "OnInitialized" event occurs when a web browser has been successfully initialized and the data connection to the PLC has been established.

Syntax

```
WebControl_OnInitialized(item)
```

Context

item

Type: Object

Web browser where the event occurs.

See also

WebControl (Page 6649)

SysDiag

Description

The "SysDiag" object enables access to the system diagnostics control.

Object type

HMIUISysDiag

Properties

--

Methods

--

See also

UI (Page 1395)

SysFct

Description

The "SysFct" object enables access to the system functions of the "SysDiag" object.

Object type

HMIUISysDiagSysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **GoToPlc()**
  Calls the "GoToPlc" method of the system diagnostics control.

See also

SysDiag (Page 6834)

SysFct.GoToPlc()

Description

The "GoToPlc" calls the "GoToPlc" method of the system diagnostics control.

Syntax

```
[HMIRuntime.]UI.SysDiag.SysFct.GoToPlc(screenItemPath);
```

Parameters

`screenItemPath`

Type: String, HmiSystemDiagnosisControl

Path of the system diagnostics control

Return value

ErrorCode

See also

SysFct (Page 6835)

SysFct

Description

The "SysFct" object ("HMIUISysFct" type) enables access to the system functions of the "UI" object.

Object type

HMIUISysFct

Properties

--

Methods

The "SysFct" object has the following methods:

- **ChangeScreen()**
  Loads a new screen into a screen window.
- **ChangeScreenAsync()**
  Loads a new screen asynchronously into a screen window.
- **ChangeScreenAsyncByNumber()**
  Loads a new screen asynchronously into a screen window.
- **ChangeScreenByNumber()**
  Loads a new screen into a screen window.
- **ClosePopup()**
  Closes a popup window in runtime.
- **GetPropertyValue()**
  Returns the current value of a property of a screen object.
- **LogOff()**
  Logs off the current user.
- **OpenScreenByNumberInPopup()**
  Opens a screen in a popup window.
- **OpenScreenInPopup()**
  Opens a screen in a popup window.

- **SetLanguage()**
  Sets a new current runtime language.
- **SetPropertyValue()**
  Sets the current value of a property of a screen object.
- **ToggleLanguage()**
  Changes the runtime language to the next one in the list of configured languages.

See also

UI (Page 1395)

SysFct.ChangeScreen()

Description

The "ChangeScreen" method loads a new screen into a screen window.

Syntax

```
[HMIRuntime.]UI.SysFct.ChangeScreen(screenName, screenWindowPath);
```

Parameters

screenName

Type: String, HmiScreen

Name of the new screen

screenWindowPath

Type: String, HmiScreenWindow

Object path of the screen window

Return value

ErrorCode

See also

SysFct (Page 6836)

SysFct.ChangeScreenAsync()

Description

The "ChangeScreenAsync" method loads a new screen asynchronously into a screen window.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]UI.SysFct.ChangeScreenAsync(screenName,
screenWindowPath)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

screenName

Type: String, HmiScreen

Name of the new screen

screenWindowPath

Type: String, HmiScreenWindow

Object path of the screen window

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

See also

SysFct (Page 6836)

SysFct.ChangeScreenAsyncByNumber()

Description

The "ChangeScreenAsyncByNumber" method loads a new screen asynchronously into a screen window.

This method executes an asynchronous operation without blocking further script execution. To do this, the method uses a Promise object which has handlers for the successful (then()) and faulty (catch()) execution of the operation. Depending on the result, the corresponding handler of the Promise pattern is called after the operation.

Syntax

```
[HMIRuntime.]UI.SysFct.ChangeScreenAsyncByNumber(screenNumber,
screenWindowPath)
.then(function() {
    ...
})
.catch(function(errorCode) {
    ...
})
```

Parameters

screenNumber

Type: UInt32

Unique number (> 0) of the new screen

screenWindowPath

Type: String, HmiScreenWindow

Object path of the screen window

Return value

Promise

Depending on the state of the Promise object:

- Promise fulfilled (fulfilled)
  No return for the "then()" handler.
- Promise rejected (rejected)
  ErrorCode as parameter of the "catch()" handler.

Example

Load the screen with the number "6" asynchronously into the screen window "ScreenWindow_1":

Copy code

```
HMIRuntime.UI.SysFct.ChangeScreenAsyncByNumber(6,"~/ScreenWindow_1")
.then(function () {
   HMIRuntime.Trace("Screen changed!");
})
.catch((error) => {
   let errMsg = HMIRuntime.GetDetailedErrorDescription(error);
       HMIRuntime.Trace("ChangeScreen failed! Error = " + error + " " +
errMsg);
});
```

See also

SysFct (Page 6836)

SysFct.ChangeScreenByNumber()

Description

The "ChangeScreenByNumber" method loads a new screen into a screen window.

Syntax

```
[HMIRuntime.]UI.SysFct.ChangeScreenByNumber(screenNumber,
screenWindowPath);
```

Parameters

screenNumber

Type: UInt32

Unique number (> 0) of the new screen

screenWindowPath

Type: String, HmiScreenWindow

Object path of the screen window

Return value

ErrorCode

Example

Load the screen with the number "6" into the screen window "ScreenWindow_1":

Copy code

```
let err = HMIRuntime.UI.SysFct.ChangeScreenByNumber(6, "~/ScreenWindow_1");
```

See also

SysFct (Page 6836)

SysFct.ClosePopup()

Description

The "ClosePopup" method closes a popup window in runtime.

Syntax

```
[HMIRuntime.]UI.SysFct.ClosePopup(popupWindowPath);
```

Parameter

popupWindowPath

Type: String, HmiPopupScreenPath

Object path of the popup window to be closed

Return value

ErrorCode

See also

SysFct (Page 6836)

SysFct.GetPropertyValue()

Description

The "GetPropertyValue" method returns the current value of a property of a screen object.

Syntax

```
[HMIRuntime.]UI.SysFct.GetPropertyValue(screenItemPath,
screenItemPropertyName);
```

Parameters

`screenItemPath`

Type: String, HmiScreenItemBase

Screen object path

`screenItemPropertyName`

Type: String

Name of the property of the screen object

Return value

Variant

See also

SysFct (Page 6836)

SysFct.LogOff()

Description

The "LogOff" method logs off the current user.

Syntax

```
[HMIRuntime.]UI.SysFct.LogOff();
```

Parameters

--

Return value

ErrorCode

See also

SysFct (Page 6836)

SysFct.OpenScreenByNumberInPopup()

Description

The "OpenScreenByNumberInPopup" method opens a screen in a popup window.

Syntax

```
[HMIRuntime.]UI.SysFct.OpenScreenByNumberInPopup(popupWindowName,
screenNumber, toggleOpen, caption, left, top, hideCloseButton[,
parentScreenPath]);
```

Parameters

popupWindowName

Type: String, HmiPopupScreenName

Name of the popup window. The name must be unique within the parent screen.

screenNumber

Type: UInt32

Unique number (> 0) of the screen that will be loaded into the popup window.

toggleOpen

Type: Bool

Specifies the behavior of the popup window when it is reopened:

- True: If the window is open, it will be closed.
- False: If the window is open, it remains open.

caption

Type: String

Specifies the window title of the popup window.

left

Type: Int32

Specifies the window position as offset from the left-hand margin.

top

Type: Int32

Specifies the window position as offset from the top margin.

hideCloseButton

Type: Bool

Specifies whether the Close button is displayed:

- True: Close button is not displayed.
- False: Close button is displayed.

parentScreenPath

Optional, type: String, HmiParentScreen

Object path of the parent screen. If this value is left empty, the popup window is global and will not close along with the parent screen.

Return value

ErrorCode

Example

Load the screen with the number "6" into the screen window "ScreenWindow_1":

Copy code

```
let err = HMIRuntime.UI.SysFct.OpenScreenByNumberInPopup("popup", 6,
false, "popup caption", 20, 30, false, undefined);
```

See also

SysFct (Page 6836)

SysFct.OpenScreenInPopup()

Description

The "OpenScreenInPopup" method opens a screen in a popup window.

Syntax

```
[HMIRuntime.]UI.SysFct.OpenScreenInPopup(popupWindowName,
screenName, toggleOpen, caption, left, top, hideCloseButton[,
parentScreenPath]);
```

Parameters

popupWindowName

Type: String, HMIPopupScreenWindow

Name of the popup window. The name must be unique within the parent screen.

screenName

Type: String, HMIScreen

Name of the screen that will be loaded into the popup window.

toggleOpen

Type: Bool

Specifies the behavior of the popup window when it is reopened:

- True: If the window is open, it will be closed.
- False: If the window is open, it remains open.

caption

Type: String

Specifies the window title of the popup window.

left

Type: Int32

Specifies the window position as offset from the left-hand margin.

top

Type: Int32

Specifies the window position as offset from the top margin.

hideCloseButton

Type: Bool

Specifies whether the Close button is displayed:

- True: Close button is not displayed.
- False: Close button is displayed.

parentScreenPath

Optional, type: String, HmiParentScreen

Object path of the parent screen. If this value is left empty, the popup window is global and will not close along with the parent screen.

Return value

ErrorCode

See also

SysFct (Page 6836)

SysFct.SetLanguage()

Description

The "SetLanguage" method specifies a new current runtime language.

Syntax

```
[HMIRuntime.]UI.SysFct.SetLanguage(lcid);
```

Parameters

`lcid`

Type: UInt32, HMILCID

ID of the new language

Return value

ErrorCode

Locale IDs (HMILCID)

The following table contains the Microsoft locale IDs of the languages supported in runtime:

Language	Country/Region	Locale ID
Afrikaans	South Africa	1078
Albanian	Albania	1052
Armenian	Armenia	1067
Azerbaijani (Cyrillic)	Azerbaijan	2092
Azerbaijani (Latin)	Azerbaijan	1068
Basque	Basque country	1069
Belarusian	Belarus	1059
Bulgarian	Bulgaria	1026
Chinese	Hong Kong S.A.R.	3076
Chinese	Macao S.A.R.	5124
Chinese	Singapore	4100
Chinese	Taiwan	1028
Chinese	PR China	2052
Danish	Denmark	1030
German	Germany	1031
German	Liechtenstein	5127
German	Luxembourg	4103
German	Austria	3079
German	Switzerland	2055
English	Australia	3081
English	Belize	10249
English	United Kingdom	2057
English	Ireland	6153
English	Jamaica	8201
English	Canada	4105
English	Caribbean	9225
English	New Zealand	5129
English	Philippines	13321
English	Zimbabwe	12297

Language	Country/Region	Locale ID
English	South Africa	7177
English	Trinidad and Tobago	11273
English	USA	1033
Estonian	Estonia	1061
Faroese	Faroe Islands	1080
Finnish	Finland	1035
French	Belgium	2060
French	France	1036
French	Canada	3084
French	Luxembourg	5132
French	Monaco	6156
French	Switzerland	4108
Galician	Galicia	1110
Georgian	Georgia	1079
Greek	Greece	1032
Hindi	India	1081
Indonesian	Indonesia	1057
Icelandic	Iceland	1039
Italian	Italy	1040
Italian	Switzerland	2064
Japanese	Japan	1041
Kazakh	Kazakhstan	1087
Catalan	Catalonia	1027
Kyrgyz	Kyrgyzstan	1088
Konkani	India	1111
Korean	Korea	1042
Croatian	Croatia	1050
Latvian	Latvia	1062
Malay	Brunei Darussalam	2110
Malay	Malaysia	1086
Macedonian	Macedonia, FYRM	1071
Mongolian (Cyrillic)	Mongolia	1104
Dutch	Belgium	2067
Dutch	Netherlands	1043
Norwegian (Bokmål)	Norway	1044
Norwegian (Nynorsk)	Norway	2068
Polish	Poland	1045
Portuguese	Brazil	1046
Portuguese	Portugal	2070
Romanian	Romania	1048
Russian	Russia	1049
Sanskrit	India	1103
Swedish	Finland	2077

Language	Country/Region	Locale ID
Swedish	Sweden	1053
Serbian (Cyrillic)	Serbia and Montenegro (formerly)	3098
Serbian (Latin)	Serbia and Montenegro (formerly)	2074
Slovakian	Slovakia	1051
Slovenian	Slovenia	1060
Spanish	Argentina	11274
Spanish	Bolivia	16394
Spanish	Chile	13322
Spanish	Costa Rica	5130
Spanish	Dominican Republic	7178
Spanish	Ecuador	12298
Spanish	El Salvador	17418
Spanish	Guatemala	4106
Spanish	Honduras	18442
Spanish	Colombia	9226
Spanish	Mexico	2058
Spanish	Nicaragua	19466
Spanish	Panama	6154
Spanish	Paraguay	15370
Spanish	Peru	10250
Spanish	Puerto Rico	20490
Spanish	Spain	1034
Spanish	Uruguay	14346
Spanish	Venezuela	8202
Swahili	Kenya	1089
Tatar	Russia	1092
Thai	Thailand	1054
Czech	Czech Republic	1029
Turkish	Turkey	1055
Ukrainian	Ukraine	1058
Hungarian	Hungary	1038
Uzbek (Cyrillic)	Uzbekistan	2115
Uzbek (Latin)	Uzbekistan	1091
Vietnamese	Vietnam	1066

See also

SysFct (Page 6836)

SysFct.SetPropertyValue()

Description

The "SetPropertyValue" method sets the current value of a property of a screen object.

Syntax

```
[HMIRuntime.]UI.SysFct.SetPropertyValue(screenItemPath,
screenItemPropertyName, value);
```

Parameters

screenItemPath

Type: String, HmiScreenItemBase

Object path of the screen object

screenItemPropertyName

Type: String

Name of the property of the screen object

value

Type: Variant

New value of the property of the screen object

Return value

ErrorCode

See also

SysFct (Page 6836)

SysFct.ToggleLanguage()

Description

The "ToggleLanguage" method toggles the runtime language to the next in the list of configured languages.

Syntax

```
[HMIRuntime.]UI.SysFct.ToggleLanguage();
```

Parameters

--

Return value

ErrorCode

See also

SysFct (Page 6836)

10.2.2.21 UserManagement

Description

The "UserManagement" object allows the roles of the current user to be checked.

Object type

HMIUserManagement

Properties

--

Methods

The "UserManagement" object has the following methods:

- **GetRolesFromUser()**
  Returns the roles of the current user.
- **HasUserRole()**
  returns whether the current user has the selected role.

UserManagement.GetRolesFromUser()

Description

The "GetRolesFromUser" method returns the roles of the current user.

Syntax

```
HMIRuntime.UserManagement.GetRolesFromUser();
```

Parameters

--

Return value

String[], HMIUserRole[]

See also

UserManagement (Page 6850)

UserManagement.HasUserRole()

Description

The "HasUserRole" method returns whether the current user has the selected role.

Syntax

```
HMIRuntime.UserManagement.HasUserRole(RoleName);
```

Parameters

`RoleName`

Type: String, HMIUserRole

Name of the role.

Return value

Bool

See also

UserManagement (Page 6850)

Planning tasks 11

11.1 Basics

11.1.1 Field of application of the Scheduler

In the Scheduler, you configure tasks that are executed in the background regardless of the screen. You create tasks by linking scripts to a trigger. The linked functions will be called when the triggering event occurs.

Application example

You use Scheduled tasks to execute event-controlled tasks automatically. For example, you use a task to automate the following:

- Regular swap out of log data
- Printout of an alarm report when an alarm buffer overflow occurs
- Printout of a report at shift end
- Monitoring a tag
- Monitoring of a user change

Note

The availability of the listed examples is dependent on the HMI device.

11.1.2 Basic of the scheduler

Definition

You use Scheduled tasks to configure tasks which are only to be executed cyclically or at a specific condition. Each task has a trigger and an action.

Triggers

You use the triggers to define when and how often the task is to be processed during runtime. The following triggers are supported:

Triggers	Type	Description
Time	Cyclic	Executed cyclically at the set time from runtime start, for example, every 2 seconds with "T2s".
Daily Weekly Monthly Yearly	Cyclic	Is executed in cycles starting from runtime start, in each case at the configured time, for example "Daily, 12:00:00 h".
Once	Acyclic	Is executed exactly once at the configured time.
Tags	Acyclic	Executed when the value of one of the projected tags changes.
Alarms	Acyclic	Executed when the state of one of the following alarm properties changes: • Alarm class, for example "Warning" • Alarm state, for example "Incoming" • Priority, for example "4"

Note

Local session tags cannot be used as triggers.

Trigger an action

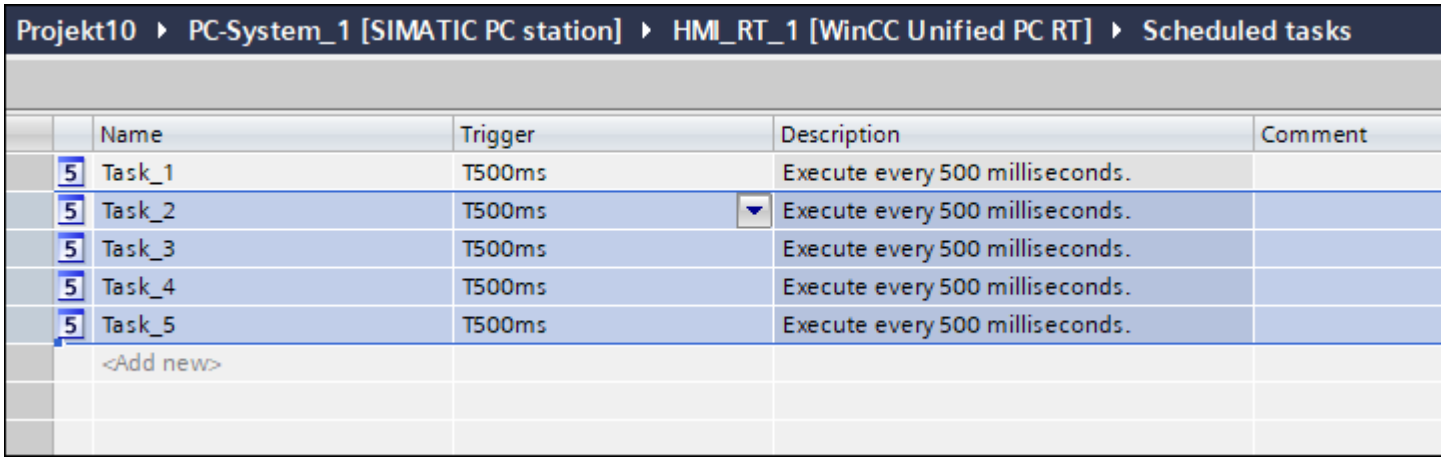
If the configured trigger condition is fulfilled, the event "Update" is triggered. You configure a local script, which triggers one or more actions.

Filling out property values of one or more tasks automatically

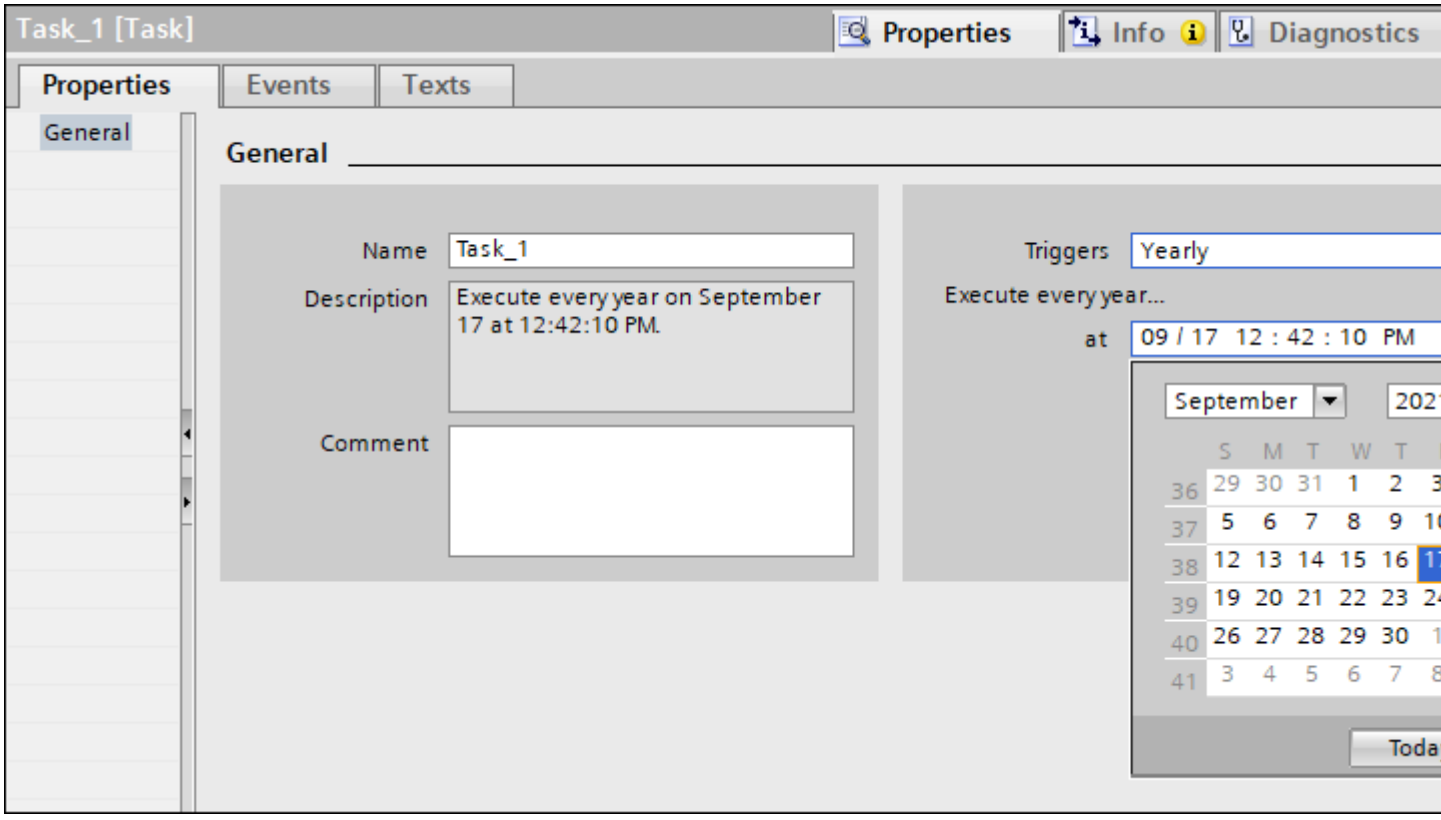
When planning new tasks, Scheduled tasks allows the property values of already defined tasks to be used. Bulk creation of property values is possible in the Scheduled tasks editor. This functionality saves the time that would be needed to create individual tasks.

Follow these steps to fill out the property values of one or more tasks automatically:

1. In the Task editor, select the "Name" or "Trigger" cell.
2. Use the mouse to drag the bottom right-hand corner downwards. The values are transferred to the destination cells.

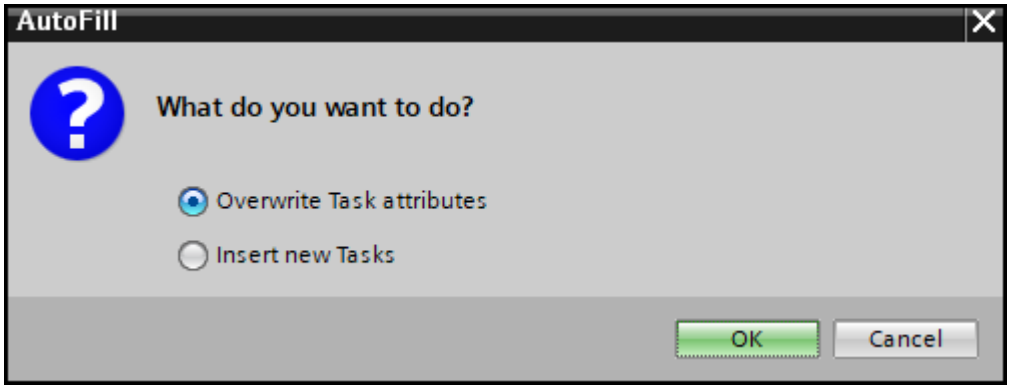


3. To change the attributes, use the text boxes in the editor or the text boxes under "Properties > Properties > General". In the latter, you can also define the time specifications, for example, time, day, month and year.



Overwriting property values of one or more tasks automatically

1. In the "Trigger" column, select the cell whose value you want to change. Select the desired value from the list.
2. To overwrite the changed value in tasks below, select the source cell in the "Trigger" column.
3. Use the mouse to drag the bottom right-hand corner downwards.
4. In the dialog box, confirm your choice with OK. The values are overwritten in the destination cells.



Sorting values in the Scheduler editor

By clicking the column header, the table is sorted alphabetically in the selected column.

See also

Creating tasks with the "Time" trigger (Page 6856)

Creating tasks with the "Tags" trigger (Page 6857)

Creating tasks with the "Alarms" trigger (Page 6857)

11.2 Creating tasks with the "Time" trigger

Requirement

- The "Scheduler" editor is open.

Procedure

Follow these steps to create a task with the trigger "Time":

1. Create a new task with "Add".
2. Select the required cycle as the "Trigger", for example "T250ms" for 250 ms.

Result

The task with the "Time" trigger has been created.

See also

Basic of the scheduler (Page 6853)
Creating tasks with the "Tags" trigger (Page 6857)
Creating tasks with the "Alarms" trigger (Page 6857)

11.3 Creating tasks with the "Tags" trigger

Requirement

- The "Scheduler" editor is open.
- You have created a tag that is monitored for changes in value.

Procedure

Follow these steps to create a task with the trigger "Tags":

1. Create a new task with "Add".
2. Select the option "Tags" as the "Trigger."
3. Select "Properties > Properties > General" in the Inspector window to select the tag.

Result

The task with the "Tags" trigger has been created.

See also

Basic of the scheduler (Page 6853)
Creating tasks with the "Time" trigger (Page 6856)
Creating tasks with the "Alarms" trigger (Page 6857)

11.4 Creating tasks with the "Alarms" trigger

Requirement

- The "Scheduler" editor is open.

Procedure

Follow these steps to create a task with the trigger "Alarms":

1. Create a new task with "Add".
2. Select the option "Alarms" as the "Trigger".
3. Configure the trigger under "Properties > Properties > General" in the Inspector window.
   - Select the "Criterion", for example "Alarm class".
   - Select the "Condition", for example "Not equal".
   - Select the "Operand", for example "Alarm".

Result

The task with the "Alarms" trigger has been created.

See also

Basic of the scheduler (Page 6853)
Creating tasks with the "Time" trigger (Page 6856)
Creating tasks with the "Tags" trigger (Page 6857)

Using the diagnostics functions 12

12.1 Configuring system diagnostics objects

12.1.1 Activating system diagnostics (S7-1200/1500)

Introduction

An integrated HMI connection must exist between the controller and the HMI device so that the controller can send error messages to the system diagnostics view. In addition, the system diagnostics must be enabled both in the PLC and on the HMI device.

With an HMI connection to a SIMATIC S7-1200/1500 controller, the system diagnostics on the HMI device can be activated and deactivated by default.

The following describes how to activate system diagnostics in a controller and on an HMI device, if necessary.

Note

System diagnostics works only for integrated connections.

Requirement

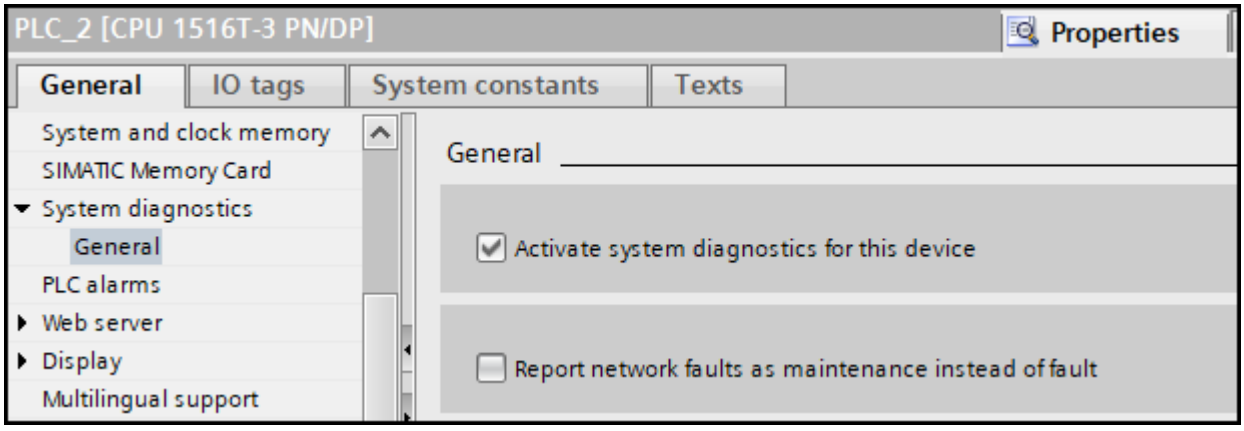
- There is an integrated HMI connection between the S7-1200/1500 controller and the HMI device (WinCC Unified PC or WinCC Unified Panel).

Activating system diagnostics in the controller

To activate the system diagnostics in a controller, proceed as follows:

1. Open the "Device configuration" of the controller in the project tree.
2. In the "Device view" tab, select the CPU on the rack.
3. Select "Properties > General > System diagnostics" in the Inspector window.

4. Activate the option "Activate system diagnostics for this device".



5. Right-click the controller in the project tree and select "Compile > Hardware (rebuild all)" in the shortcut menu.

Activating system diagnostics on the HMI device

To activate the system diagnostics on an HMI device, proceed as follows:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Select the option "System diagnostics" under "Alarms > Controller alarms".
   The display of system diagnostic alarms is enabled in Runtime.

Note

When an upload is performed from the PLC to the TIA Portal, the uploaded PLC must be compiled to HMI RT before it is compiled and downloaded.

This is system behavior, because the runtime data file is created during compiling.

12.1.2 Configuring diagnostics indicators (S7-1200/1500)

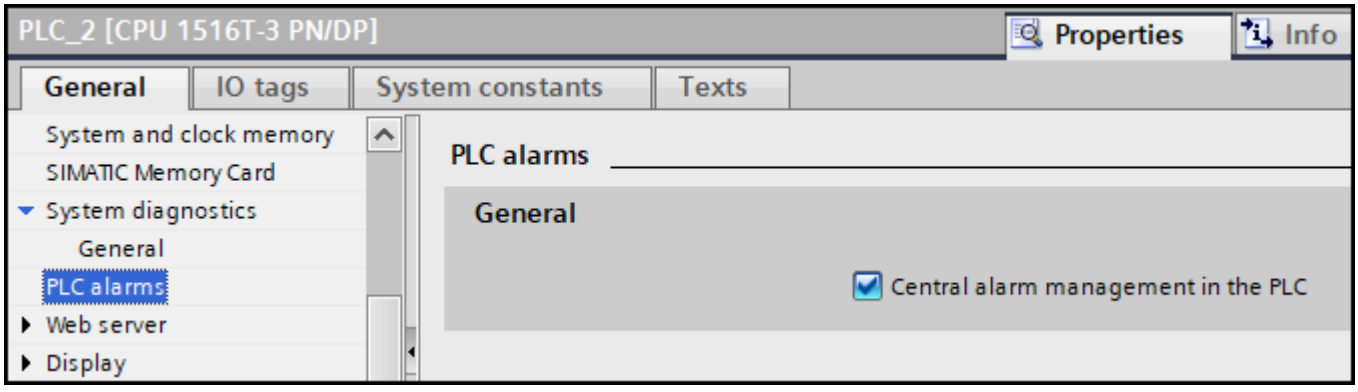
Showing the overall status of HMI connections via traffic light SVGs

The diagnostic status is represented by a system tag named "@DiagnosticsIndicatorTag". The "System" is notified of the diagnostic status of the configured PLCs.

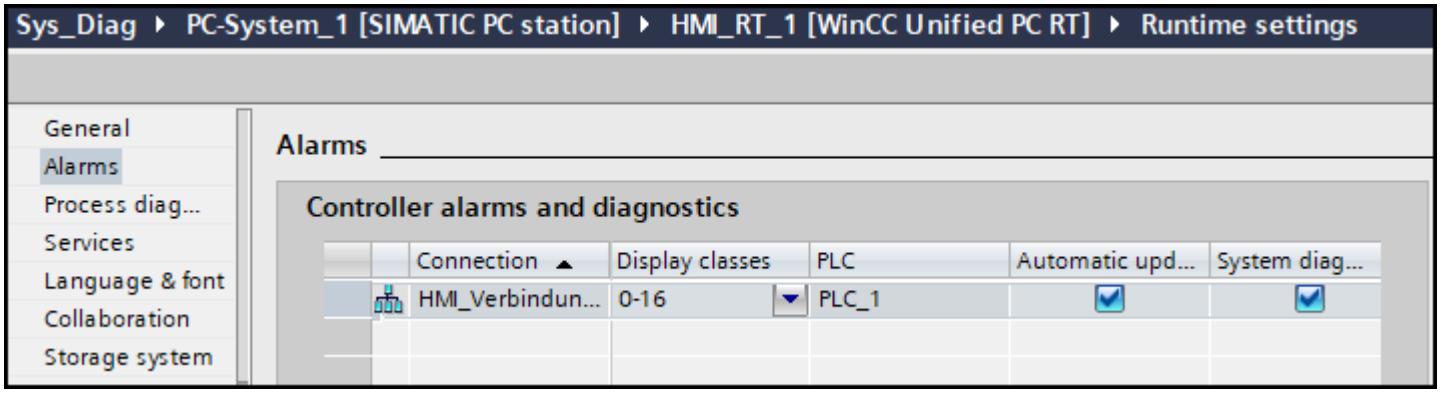
The diagnostic status contains the overall status of all relevant PLCs. The merged state always corresponds to the worst state of all relevant PLCs.

Requirements

- Integrated HMI connection with PLC S7-1200/1500.
- PLC setting "Central alarm management in the PLC" is enabled.



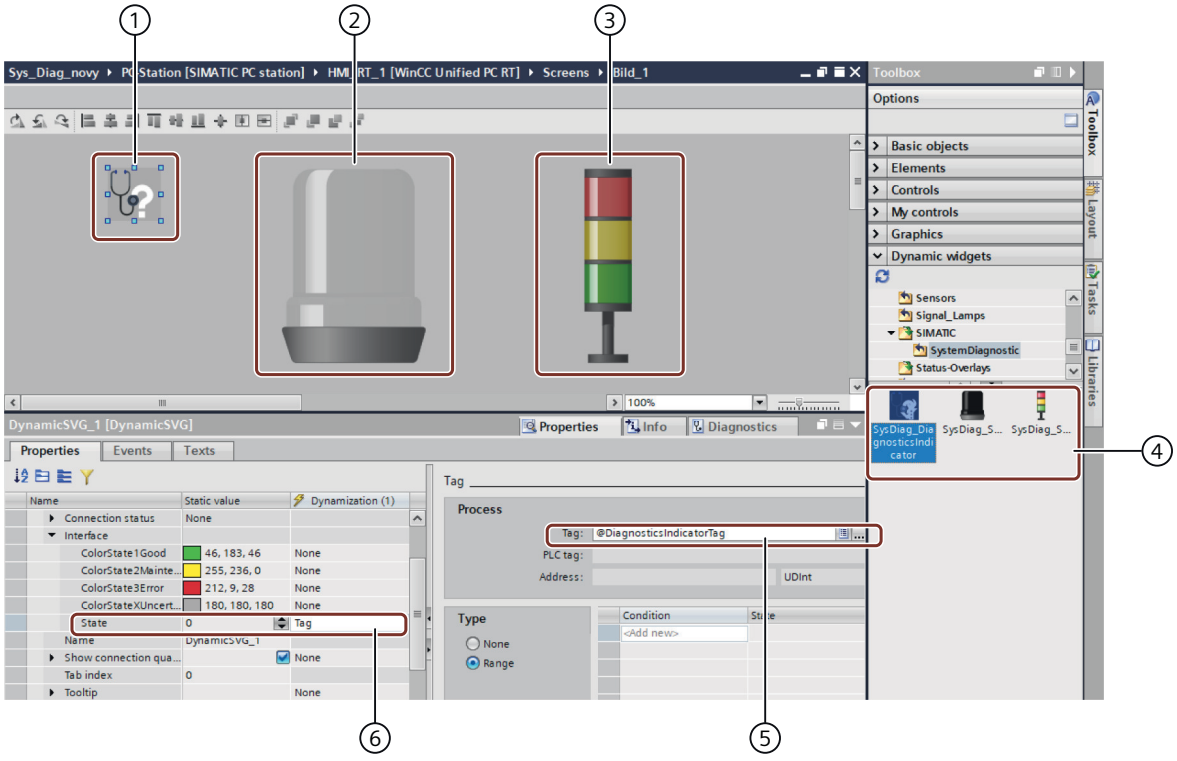
- "Automatic update" and "System diagnostics" are enabled for the controller alarms of the HMI connection.



Using dynamic SVGs

System diagnostics for RT Unified provides 3 pre-programmed dynamic SVGs for S7-1200/1500 PLCs as tools for dynamic widgets:

- Diagnostics indicator ①
- Signal lamp ②
- Signal tower ③



To create a diagnostics indicator, proceed as follows:

1. Open the "Toolbox" and find "IndustryGraphicLibrary > Dynamic widgets > SIMATIC > SystemDiagnostic" ④.
2. Select one of the 3 pre-programmed SVGs:
   - "SysDiag_DiagnosticsIndicator"
   - "SysDiag_SignalLamp"
   - "SysDiag_SignalTower"
3. Double-click or drag-and-drop the selected SVG over to the screen. The object is added to the screen.
4. Open the properties of the object in the Inspector window. Assign the "Tag" dynamization to the interface element "State" ⑥.
5. Dynamize the State property with the tag "@DiagnosticsIndicatorTag" ⑤.

The color of the dynamized SVG changes according to the defined tag values in Runtime.

Table 12-1 Possible diagnostic values

Status	Diagnostic value	Color
Uncertain	0	Gray
Good	1	Green
Maintenance	2	Yellow
Error	3	Red

12.1.3 Configuring system diagnostics of the controller (S7-1200/1500)

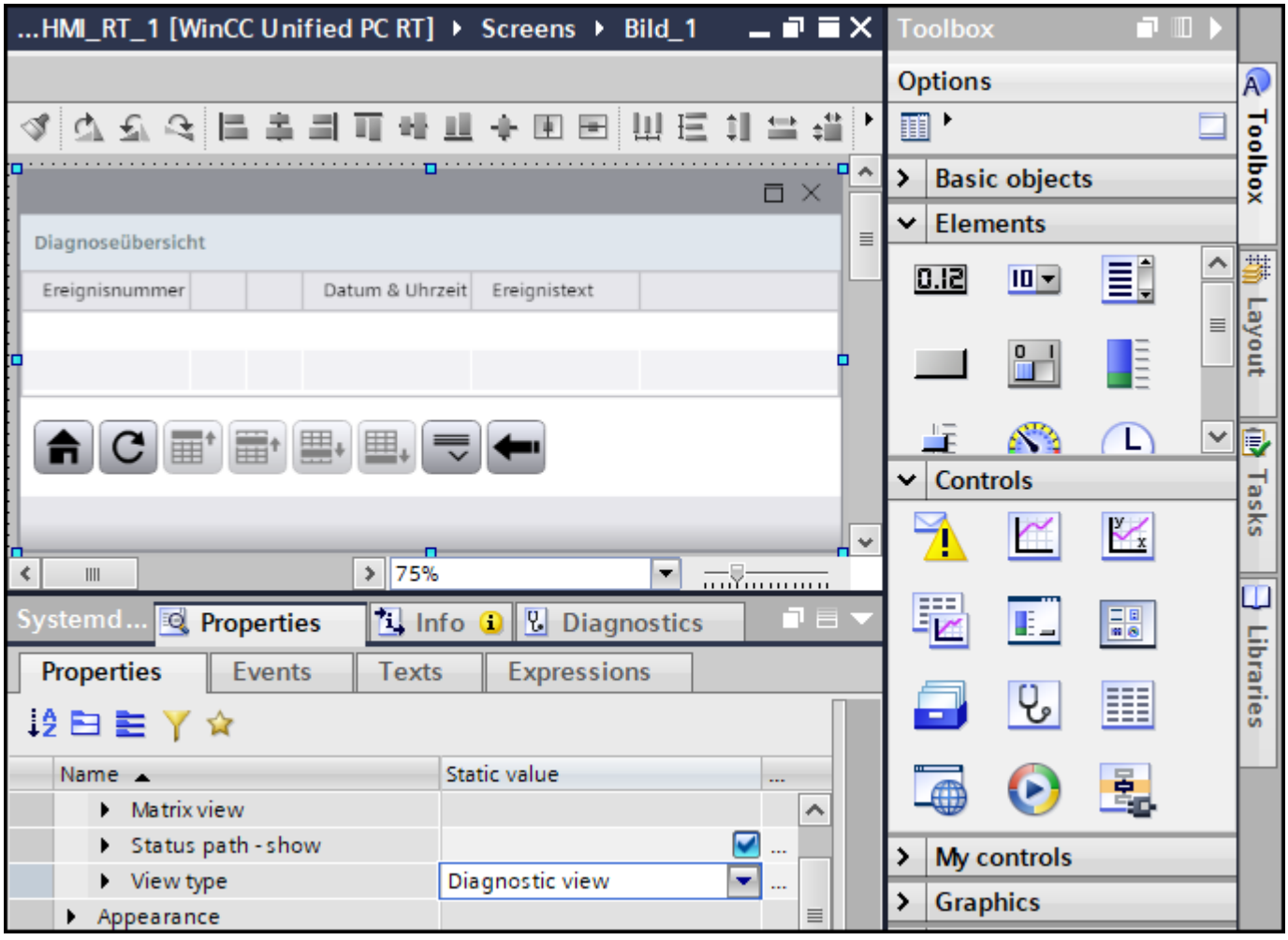
The system diagnostics control shows the diagnostic events of the configured PLCs in your HMI device. When loading the screen, the control displays the diagnostic buffer of the PLC with the most serious status. Navigation buttons can be used to navigate to the next PLC.

Requirements

- At least one S7-1200/1500 PLC is configured. The S7-1200 PLC is supported as of firmware version 4.0.
- PLC setting "Central alarm management in the PLC" is enabled.
- An HMI device has been configured.
- An integrated HMI connection has been established between the S7-1200/1500 PLC and the HMI device.
- "Automatic update" and "System diagnostics" are enabled for the controller alarms of the HMI connection.
- System diagnostics is enabled in every controller and on the HMI device.
- A screen has been created.
- The Inspector window is open.

Procedure

Double-click the icon of the "System diagnostics control" object in the "Controls" section of the "Toolbox" task card, or drag it into the screen using drag-and-drop. The object is added to the screen.



You can change the setting for the position, geometry, style, color, and font of the object in the Inspector window. You can adapt the following properties in particular:

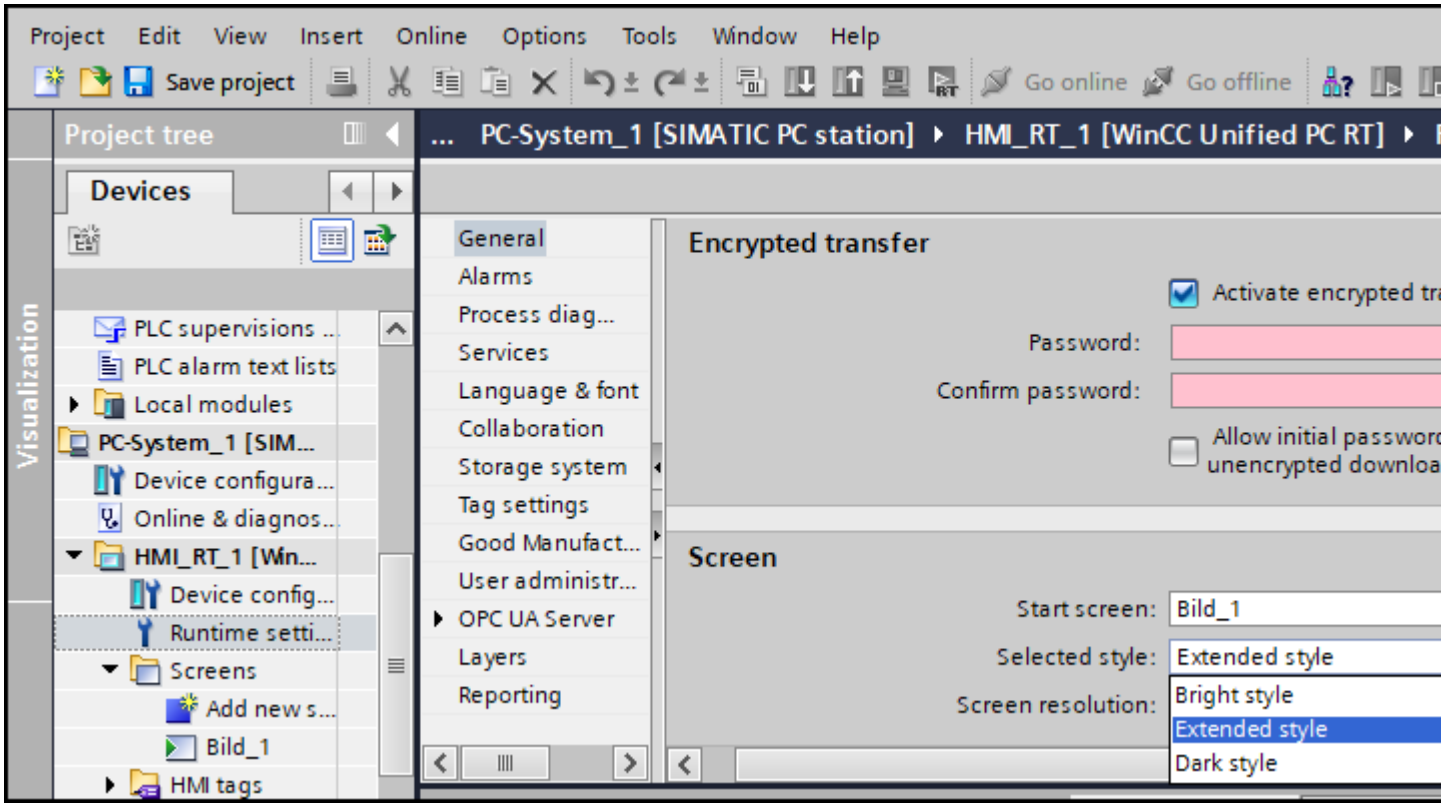
- "View type": Switches between the diagnostic view and the matrix view.
- "Information bar": Specifies the representation of the information bar.
- "Toolbar": Specifies the buttons of the system diagnostics control.

Setting pre-defined styles

You can assign pre-defined styles to the object in Runtime: "Dark Style", "Extended Style" or "Bright Style". You can use pre-defined styles to change the background color of the object.

To assign a style, follow these steps:

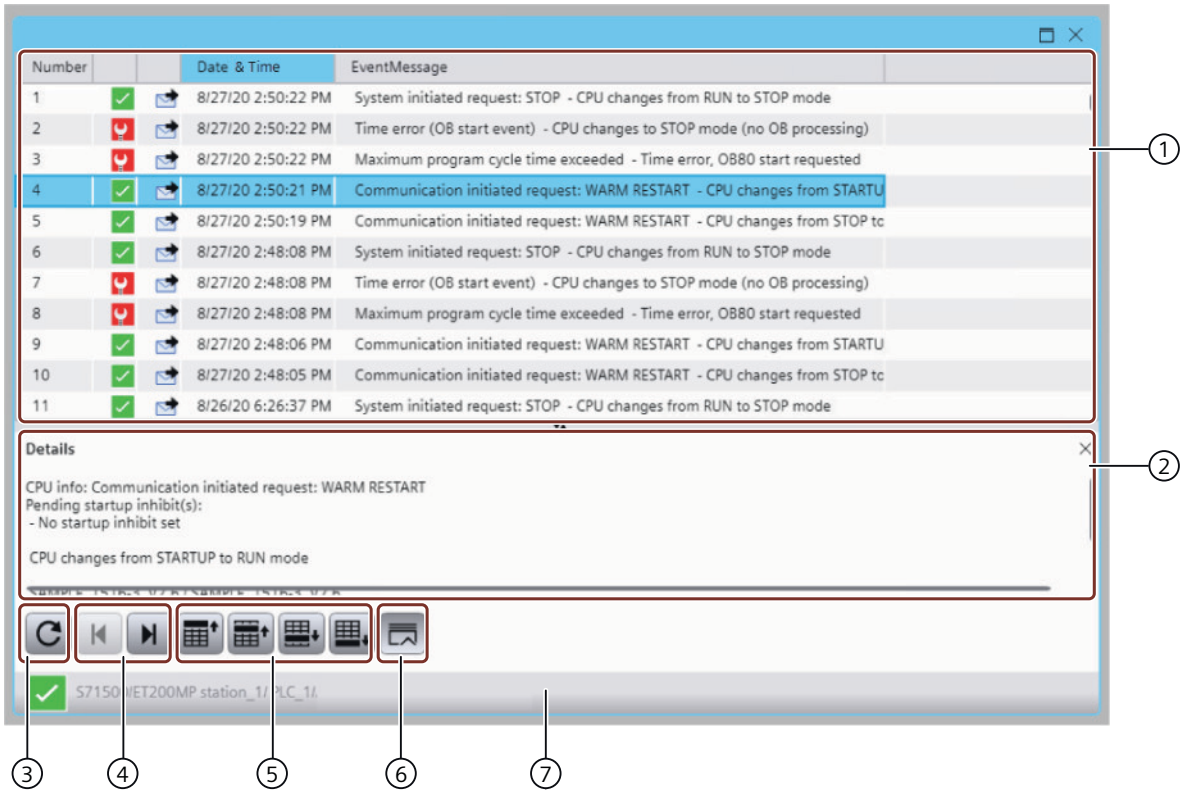
1. Open the runtime settings of the HMI device.



2. In the "General > Screen > Selected style" tab, select one of the following options: "Bright style", "Extended style" or "Dark style" to use the style.

Result

The "System diagnostics control" has been added to the screen. The system diagnostics can access the data of the integrated HMI connections to S7-1200/1500 PLCs. In Runtime, the diagnostic alarms of the selected PLC are displayed in the "System diagnostics control". The selected PLC can be changed using the buttons ④. Once Runtime has started, the events of the PLC with the most serious error are displayed.



① Grid view
② Detail view
③ Update the view of the diagnostics event
④ Switch to the next or previous PLC
⑤ Navigation buttons for the grid view:
jump to the first line
jump to the previous line
jump to the next line
jump to the last line
⑥ Enable/disable detail view
⑦ Status text field

The diagnostic buffer displays the diagnostics events of a PLC in a grid view ①. The grid view shows the last 200 diagnostics events of the PLC.

The first column shows the number of the entry.

The symbols in the second column indicate the event type of the PLC:

	Device in operation
	Maintenance required
	Maintenance necessary
	Error in the device

You can see the symbols of the incoming or outgoing status in the third column:

	Incoming event
	Outgoing event
	Incoming event for which there is no independent outgoing event
	User-defined diagnostics event

The fourth column shows the date and time of the event. You can see the event information in the last column.

Below the grid view, the detail view ② of the selected row from the grid view is displayed. You can enable or disable the detail view with the button ⑥.

When the screen is loaded, the "System diagnostics control" shows the PLC with the most severe error. If several PLCs are configured for system diagnostics, you can use the toolbar buttons ④ to switch to the next or previous PLC.

To update the "System diagnostics control", select the toolbar button ③. For performance reasons, no automatic update is performed. You need to perform the update manually.

At the bottom of the window, a status field ⑦ is displayed with the diagnostic status and the name of the station/PLC.

Note

Rearranging columns

You have the option of changing the column order configured in the engineering system. You can find more detailed information in the "AUTOHOTSPOT" manual in the section "Rearranging columns at runtime (Page 7451)".

Languages in Runtime

The alarms are displayed in the RT language selected by the user in the screen logon dialog. The Runtime language and the PLC language should be identical.

The PLC supports only three languages, which can be configured by the user in the engineering. If the PLC language and the runtime language are different, the event text is displayed as follows according to the fallback mechanism:

- English US
- English UK
- the standard text "## text is missing ##"

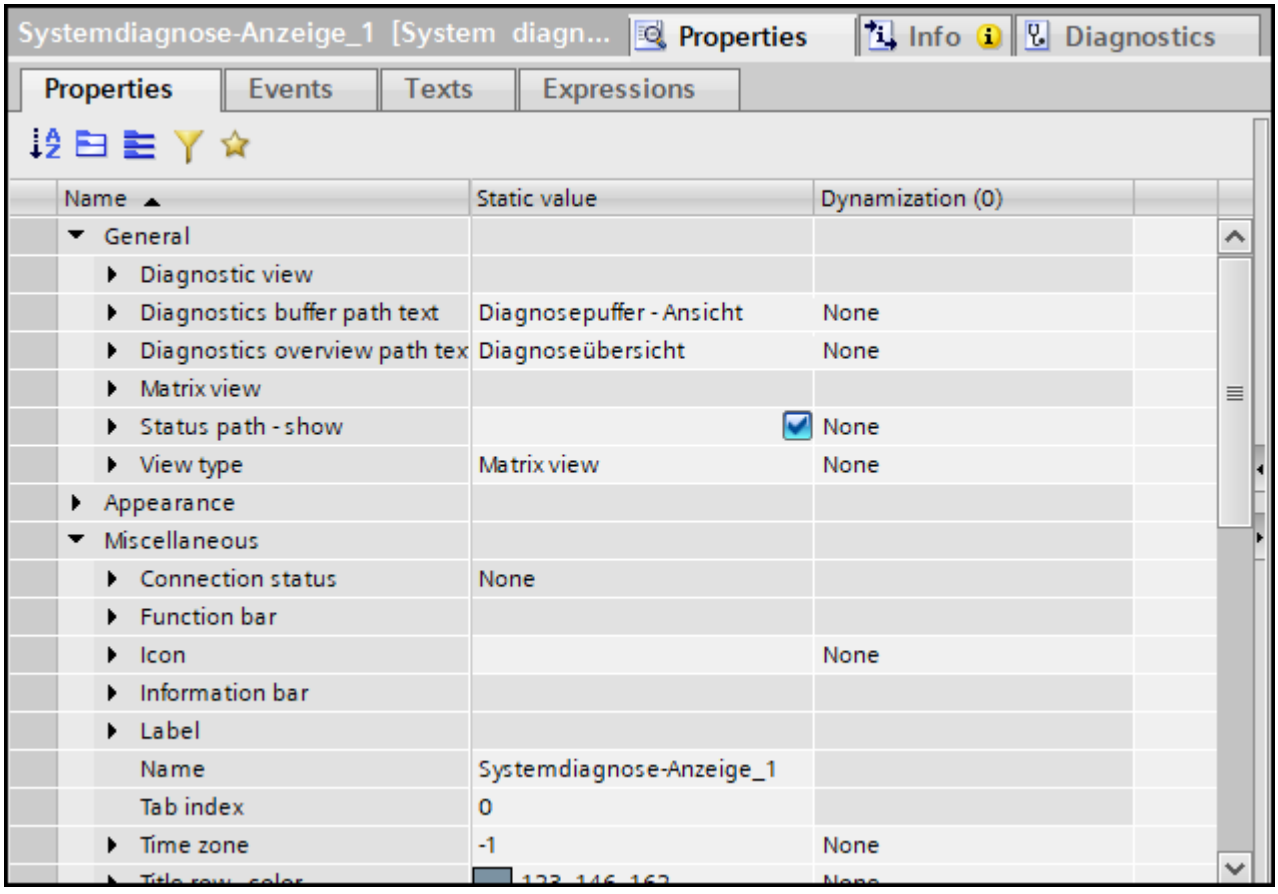
12.1.4 System diagnostics display

Use

You can use the "System diagnostics control" object to display the diagnostic status of several PLCs using traffic light SVGs. The diagnostic status contains the overall status of all relevant PLCs. The merged state is always the worst state of all PLCs.

Defining the properties of the system diagnostics control

You define the properties of the system diagnostics control in the Inspector window under "Properties > Properties".



Selecting the view type

You select the view type in the following way:

1. Click "Properties > Properties > General > View type" in the Inspector window.
2. Choose between the "Matrix view" and the "Diagnostic view".

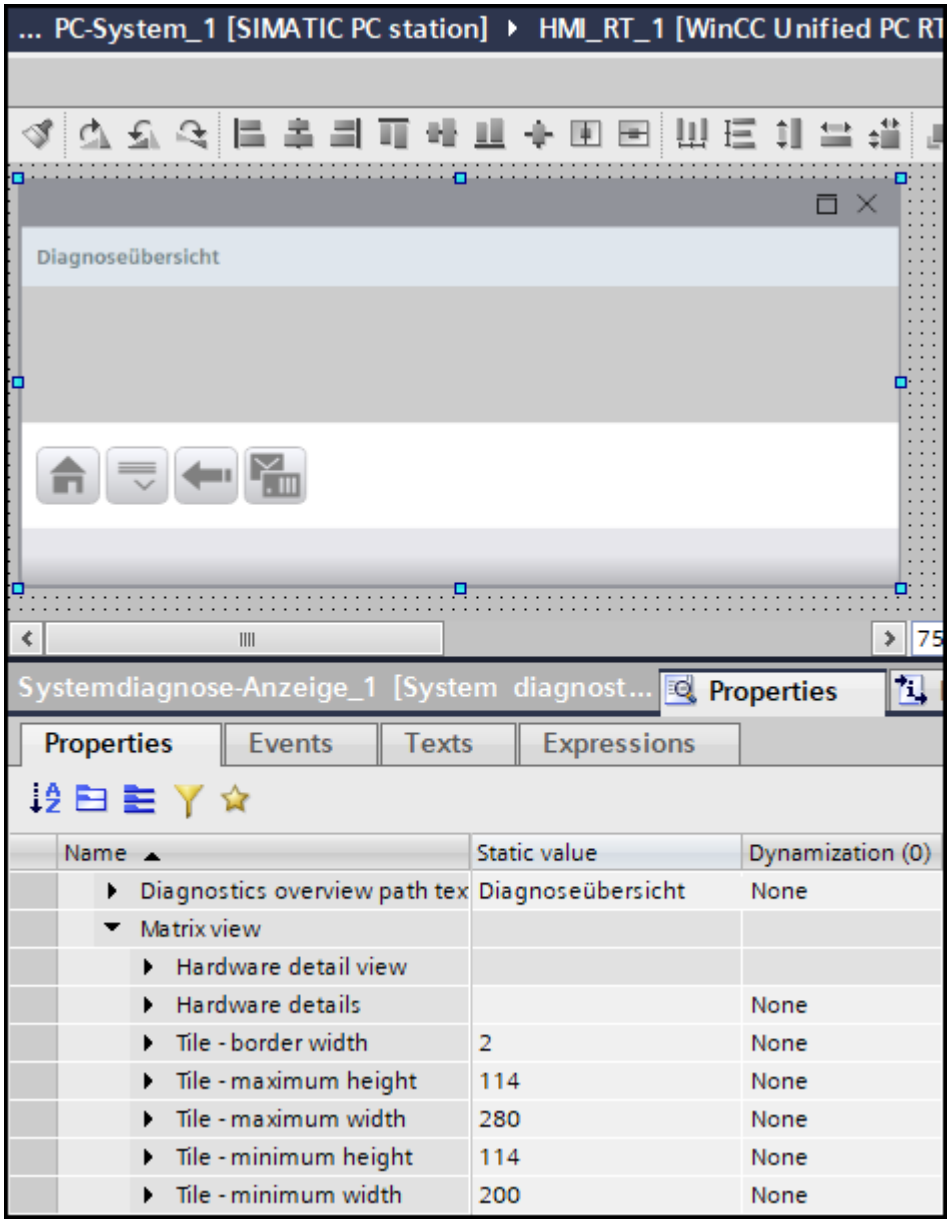
Selection of the matrix view as start view is recommended. From the matrix view, you can switch to the diagnostic view using the corresponding button in the toolbar.

Matrix view

With the matrix view, you have the possibility to check the status of your PLCs and their lower-level hardware components.

All hardware components are displayed as tiles. You can configure the display as well as the content of the tiles:

Make the settings for hardware details and tiles under "Properties > Properties > General > Matrix view".

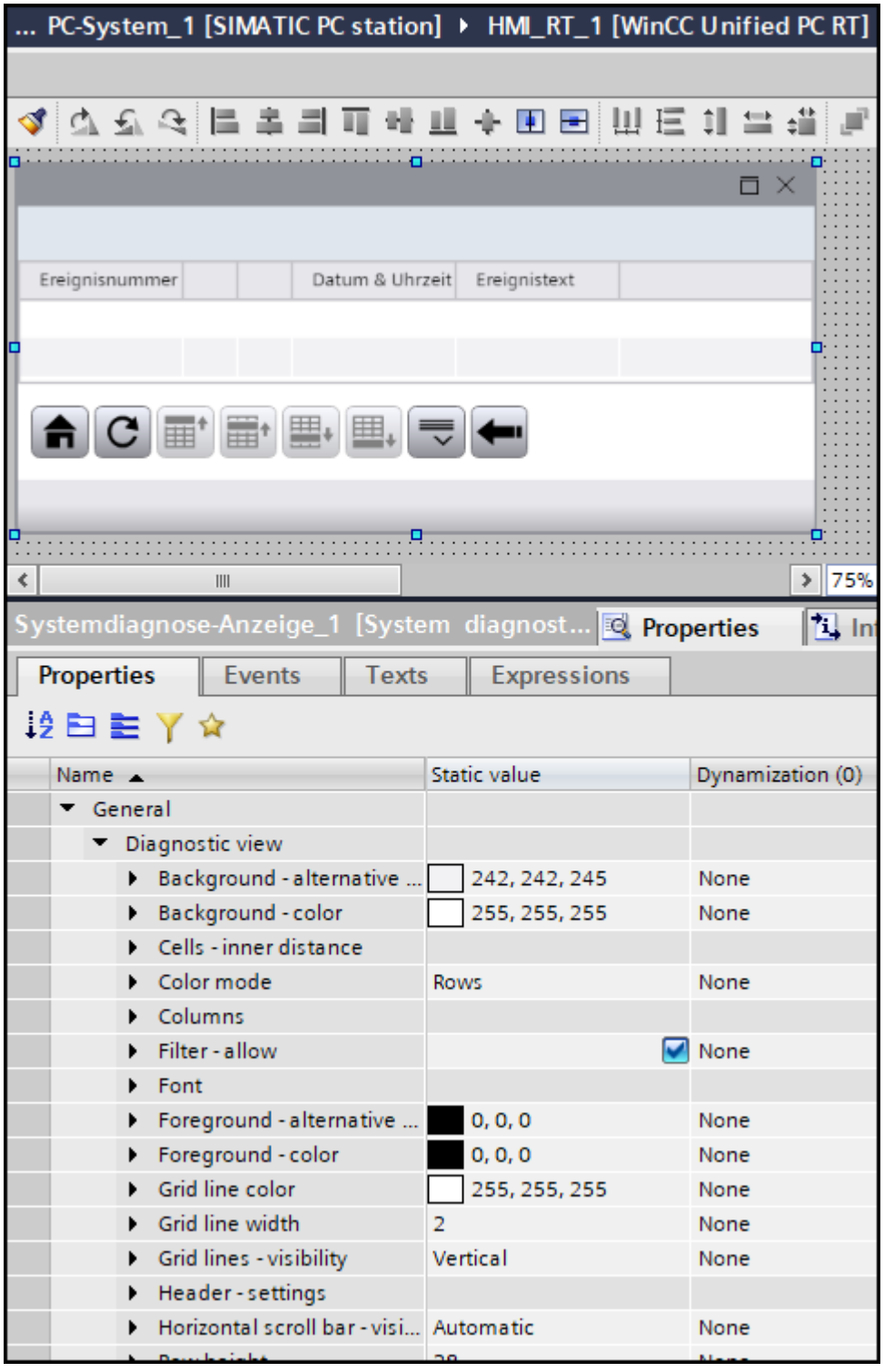


Diagnostic view

The diagnostic view shows the diagnostic buffer of the PLC with the diagnostic events of the currently selected PLC.

It is not possible to switch between different PLCs in Runtime. Navigating to the diagnostic view via the selected PLC in the matrix view is recommended.

Under "Properties > Properties > General > Diagnostic view", you make the settings for the rows, header, grid lines, scroll bar, cells and columns.


... PC-System_1 [SIMATIC PC station] ▸ HMI_RT_1 [WinCC Unified PC RT]
Ereignisnummer
Datum & Uhrzeit
Ereignistext
75%
Systemdiagnose-Anzeige_1 [System diagnost...
Properties
Properties
Events
Texts
Expressions
Name
Static value
Dynamization (0)
General
Diagnostic view
Background - alternative ...
242, 242, 245
None
Background - color
255, 255, 255
None
Cells - inner distance
Color mode
Rows
None
Columns
Filter - allow
None
Font
Foreground - alternative ...
0, 0, 0
None
Foreground - color
0, 0, 0
None
Grid line color
255, 255, 255
None
Grid line width
2
None
Grid lines - visibility
Vertical
None
Header - settings
Horizontal scroll bar - visi...
Automatic
None

Setting up column sorting

To set up the column sorting in the diagnostic view, follow these steps:

1. In the Inspector window, click "Properties > Properties > General > Diagnostic view > Columns > [0] Column".
2. Select the sorting direction and sorting order for the individual columns.

Dynamization of graphic properties with tags or scripts

You can dynamize the following properties containing a graphic with a tag or with a script:

- Graphic
- Icon

Access protection in Runtime

Configure access protection with the property "Operator control - allow" in the Inspector window under "Properties > Properties > Security". A logged-in user having the required authorization can acknowledge and edit the system diagnostics control using the buttons in the system diagnostics control.

Configuring the information bar

The information bar of the system diagnostics control shows the connection status and path.

The connection status is not displayed while the PLC is starting.

To configure the information bar, follow these steps:

1. Configure the general properties of the information bar, such as the font and background color, under "Properties > Properties > Miscellaneous > Information bar".
2. Configure the display of the information bar elements under "Properties > Properties > Miscellaneous > Information bar > Elements".

Toolbar

You can define the buttons of the system diagnostics control in Runtime and their operator authorizations in the Inspector window under "Properties > Properties > Miscellaneous > Toolbar > Elements". Some buttons are enabled by default. To display additional buttons in the object, activate the "Visibility" property in the settings of the corresponding button.

The following buttons are available for the system diagnostics control:

Button		Function
	Home	Shows the home page.
	Reload	Updates the view of the diagnostic event.

Button		Function
	First line	Selects the first of the pending diagnostic events. The visible area of the view is moved.
	Previous line	Selects the previous diagnostic event, starting from the currently selected diagnostic event. The visible area of the view is moved.
	Next line	Selects the next diagnostic event, starting from the currently selected diagnostic event. The visible area of the view is moved.
	Last line	Selects the last of the pending diagnostic events. The visible area of the view is moved.
	Share view	Enables/disables the detail view.
	Previous	Navigates to the previous PLC.
	Show diagnostic buffer	Changes from the matrix view to the diagnostic view. The diagnostic view shows the diagnostics buffer of the PLC. This button is only enabled if a PLC or one of its lower-level modules is shown in the matrix view.

Setting the time zone

Under Properties > Properties > Miscellaneous > Time zone, you set the desired time zone by entering a decimal value for the time zone.

- "0" and positive numerical values: The values correspond to the index values of the Microsoft time zones.
- "-1": The local time zone of the device.

Note

In Runtime, you also have the option of setting the time zone via a selection list.

12.2 Example: System diagnostics with all objects

12.2.1 Example: Procedures overview

Introduction

- A controller and an HMI device have been created.
- An HMI connection has been established between the controller and HMI device.
- System diagnostics is activated in the controller and on the HMI device.

Configuration steps

To get a quick view of errors, create an overview screen with the various objects for displaying diagnostic alarms.

The following example shows how to efficiently use the objects of the "Tools" or "Libraries" task cards in your project.

The example is divided into several steps:

- Creating screens
  The project engineer creates multiple screens for system diagnostics:
  - Overview screen with all objects for system diagnostics
  - Screen for the alarm view
  - Screen for the system diagnostics view
- Inserting objects in the screens
  The project engineer inserts various objects in the screens:
  - Alarm view in the "Alarm" screen
  - Screen window for displaying the alarm view and system diagnostics view
- Configuring objects
  The project engineer connects objects to enable targeted navigation to the cause of the error in runtime.
  - System diagnostics indicator with screen window of the system diagnostics view
  - Goto button with the screen window of the alarm view and the system diagnostics view.

12.3 Process diagnostics

12.3.1 Basics of supervision with ProDiag

Introduction

The TIA Portal functionality, ProDiag (Process Diagnostics), is used to monitor and determine errors that occur in your plant or machine. You can use ProDiag to show the type of error, the cause of the error and the location of the error on the HMI device.

Use

You can use ProDiag functions to monitor your plant and to visualize it on an HMI device. The main objective of ProDiag is the reduction of downtime and loss of production after an error occurs, and the avoidance of errors using timely warnings. Diagnostic and display objects provide specific information for the operator for troubleshooting and show the processes on an HMI device on site.

Principle

In STEP 7, you create operand supervisions and configure the settings according to your requirements. When an error occurs, a supervision alarm is generated based on the criteria you have configured. The configured supervision instances are stored in the preset ProDiag function block. You can use the automatically generated ProDiag FBs or create and configure your own ProDiag FBs.

Advantages

ProDiag enables you to configure supervisions and monitor your plant without changing the user program code.

You perform plant diagnostics on your HMI device. The data is automatically synchronized in order to keep the display on your HMI device always up-to-date.

12.3.2 Requirements and licensing

Introduction

You configure the ProDiag supervisions in TIA Portal with STEP 7 and create the screen objects for monitoring and diagnostics with WinCC. You need a license to use the ProDiag functionality and the corresponding screen objects.

Software requirements

To configure ProDiag supervisions, you need the following products:

- TIA Portal STEP 7 Professional
- WinCC Unified

Hardware requirements

ProDiag functionality is available for CPUs of the S7-1500 series with firmware version 2.0 or higher.

The objects for the supervision and diagnostics of plants are available for the following HMI devices:

- WinCC Unified

Note

Objects for supervision and diagnostics of plants can be used under the "Full access" and "Read access" protection levels configured in the CPU.

ProDiag objects under the "HMI access" and "No access" protection levels cannot be used.

Licensing of ProDiag supervisions

The number of ProDiag monitors that you configure with STEP 7 is licensed. You do not need a license for the first 25 supervisions, licenses must be used for additional supervisions.

Number of supervisions	**<= 25**	**<= 250**	**<= 500**	**<= 750**	**<= 1000**	**> 1000**
Number of licenses	None	1	2	3	4	5

Licensing of ProjDiag objects

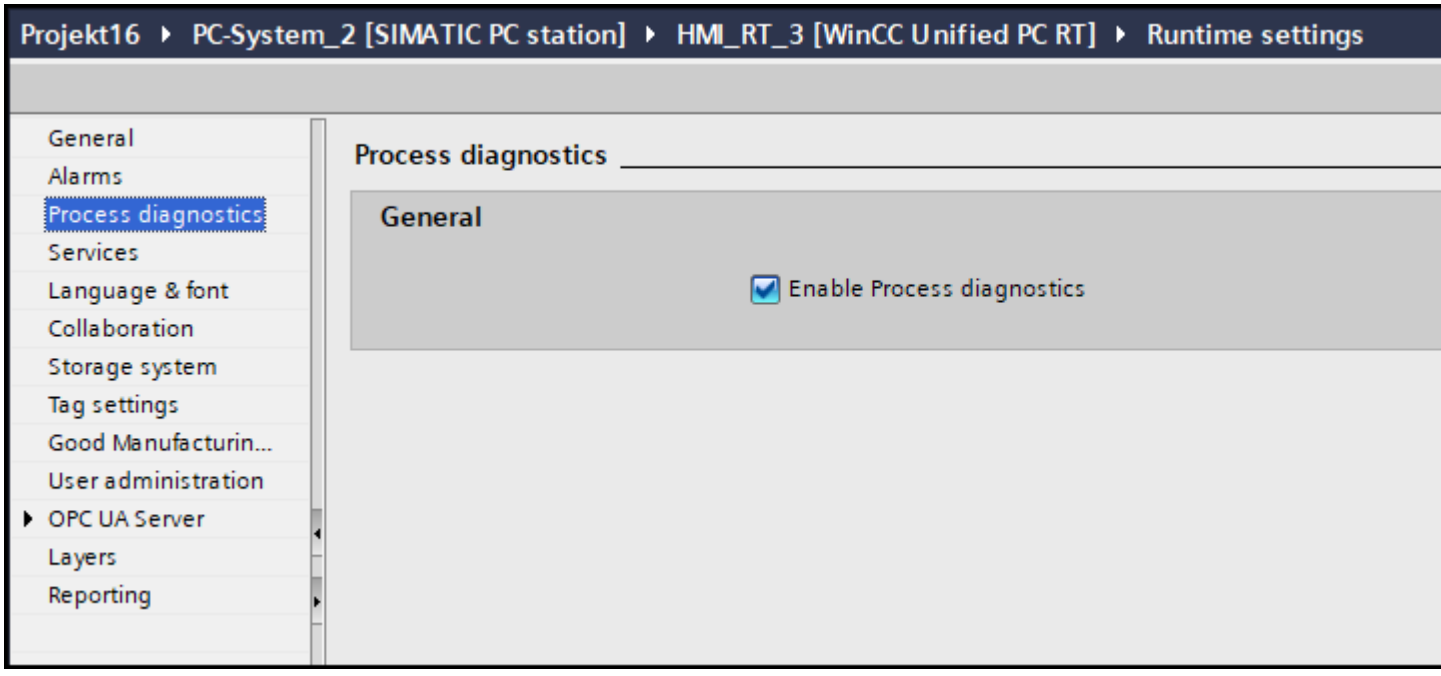
To use the objects for the diagnostics and supervision in conjunction with the ProDiag supervision in your program, you need a ProDiag license, specifically the WinCC Unified Runtime license.

Enable process diagnostics

To activate process diagnostics on an HMI device, follow these steps:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Under Process diagnostics, select the "Enable process diagnostics" option.

The display of process diagnostic alarms is enabled in runtime.



12.3.3 Objects for the supervision and diagnostics of plants

Introduction

WinCC offers the following objects for displaying the current monitoring status and for fault diagnostics in the program code:

- GRAPH overview
- PLC code view

GRAPH overview

The "GRAPH overview" object is used to display the current program status for executed steps of the GRAPH sequencer.

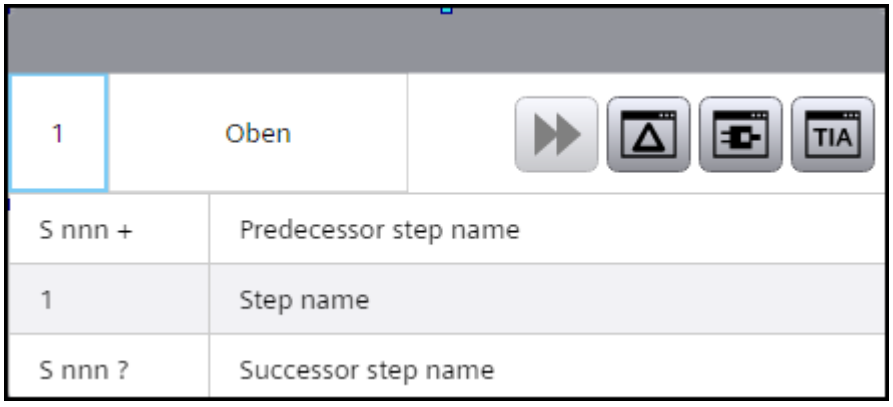
PLC code view

The "PLC code viewer" object is used to display the current program status of user programs that have been programmed in the GRAPH programming language.

12.3.4 GRAPH overview

Use

The "GRAPH Overview" object is used to display the current program status for executed steps of the GRAPH sequencer. Errors during execution of a program are displayed directly at the corresponding step.



The following information is displayed in the "GRAPH Overview" object:

- Name and status of the function block
- Status of initial and simultaneous steps
- Number and name of the first step currently executed step
- Operating mode for running the GRAPH sequencer

WinCC supports the display of step names for the GRAPH blocks in multiple languages starting from Version 6.0. The step names will then be displayed in the selected Runtime language following a language changeover in Runtime. If the selected language is not available in the GRAPH block, the names are displayed in the default language (English).

Note

Device dependency of the "GRAPH Overview" object

The "GRAPH overview" object is available for Unified PC.

Note

Requirement for display in GRAPH overview

For the display of the program status of an S7 GRAPH instance data block in the "GRAPH overview" object to be possible, the instance-specific properties of the block must be set as "Visible in HMI" and "Accessible from HMI".

Layout

In the Inspector window, you customize the position, style, colors and font types of the object. You can adapt the following properties in particular:

- Assigned GRAPH DB tag
- Buttons of the toolbar

Operating mode

Four operating modes are available for running the GRAPH sequence:

- AUTO (default setting) - Automatically switches to the next step when the transition is fulfilled.
- TAP - Automatically switches to the next step when the transition is fulfilled and there is an edge change from "0" to "1" at the T_PUSH parameter.
- TOP - Automatically switches to the next step when the transition is fulfilled or there is an edge change from "0" to "1" at the T_PUSH parameter.
- MAN - The next step is not automatically enabled when the transition is fulfilled. The steps can be selected and deselected manually.

Note

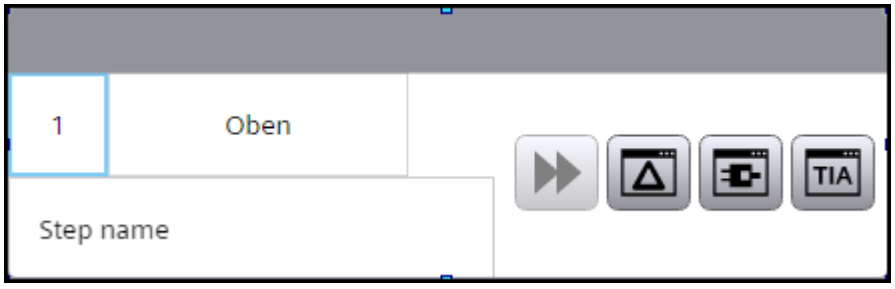
You set the operating mode by modifying the interface parameters of the GRAPH block in your control program.

In WinCC Unified Runtime, you have the option to customize the name for the operating mode that is displayed in the GRAPH overview.

Configuring a compact view

You can also configure a slim GRAPH overview without toolbar buttons and operating mode display.

To display a slim GRAPH overview in single-line compatibility mode, drag the control to the desired size.



Symbols

The symbols displayed in the GRAPH overview are pre-defined:

Symbol	Name	Function
	Error	Indicates that an error has occurred during the execution of a step.
	Initial step	Indicates that the currently executing step is the first step in the GRAPH block.
	Simultaneous step	Shows that there are other simultaneous steps in the GRAPH block in addition to the current one.

Buttons

You specify the buttons that are displayed in the GRAPH overview under "Properties > Miscellaneous > Toolbar > Elements".

Button	Name	Function
	Next Step	Jumps to the next step in parallel step. When you get to the last step, you can jump back to the first step.
	Jump to Alarm Control	Opens the configured alarm view with the error alarm in WinCC Unified. The button is intended to be populated with appropriate system functions/ scripts.
	Jump To PLC Code Viewer	Opens the configured PLC code view. The button is intended to be populated with appropriate system functions/ scripts. Ideally, use the "OpenViewerGraphFromOverview" system function.
	Jump to TIA Portal	Several system functions are available for opening the TIA Portal.

12.3.5 Configuring a GRAPH overview

Introduction

You can use the GRAPH overview to view the current program status for the executed steps of a GRAPH sequencer.

Requirement

- A PLC including a GRAPH instance data block has been created.
- GRAPH instance data block contains at least one tag which is visible in HMI and accessible from HMI.

  Note

  The process tag you are using for the GRAPH overview must be visible in HMI and accessible from HMI.

  To identify the tags of the GRAPH data block as visible and accessible for HMI, open the GRAPH function block, select the block in the work area, and select "Edit > Internal parameters visible/ accessible from HMI" in the menu bar. Then compile the program blocks.

- An HMI device has been created.
- You have created a screen.
- The Inspector window is open.

Procedure

1. Drag-and-drop the GRAPH overview from the toolbox view into the configured screen.
2. In the Inspector window, click "Properties > Properties > Miscellaneous".
3. Open the selection button under "PLC Source > Tag".
   The "Add new object" dialog opens.
4. Select the corresponding PLC in the "Program blocks" folder.
5. Select the corresponding PLC tag of the GRAPH instance data block.

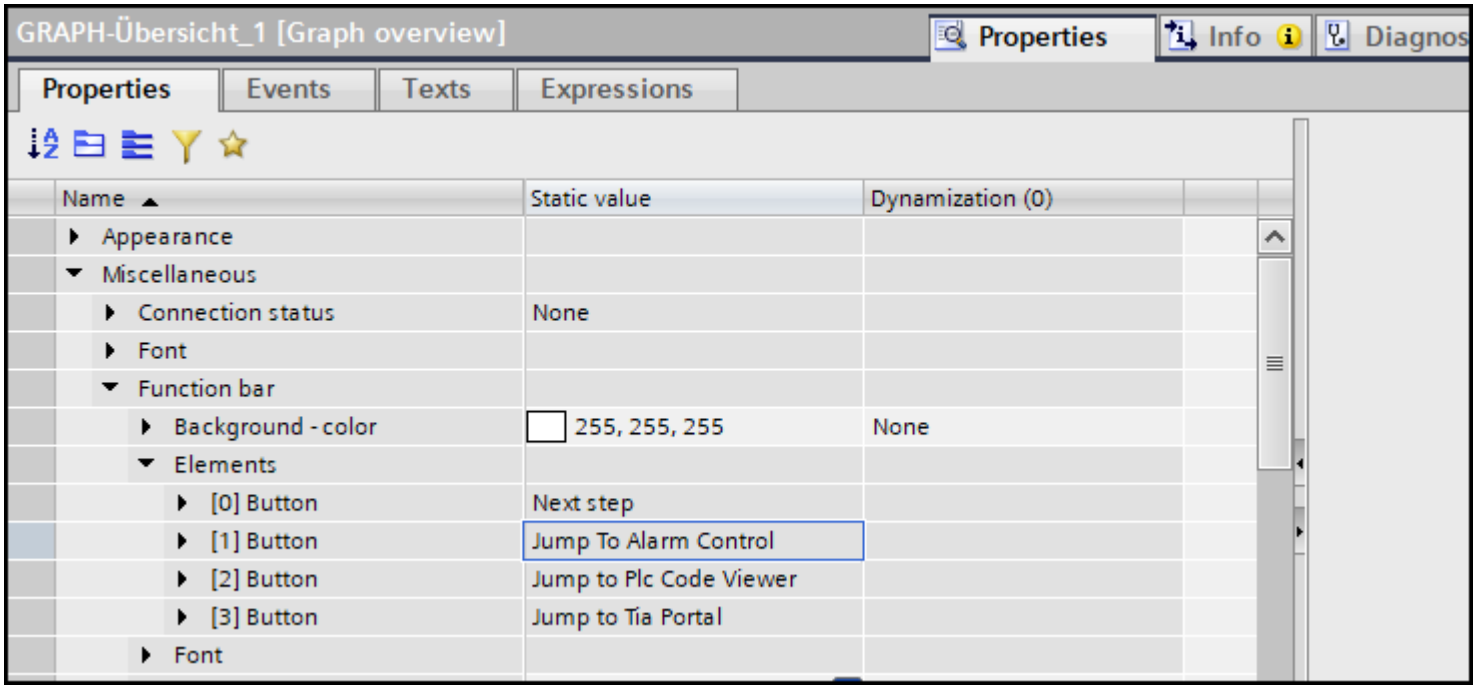
   Note

   If no connection was configured between the HMI device and the selected PLC, a connection is set up automatically.

   In addition, an HMI tag is created which is connected to the PLC tag.

6. To display the GRAPH overview in compatibility mode without toolbar buttons and operating mode display, drag the control to the desired size, whereby several basic views are possible.

7. Under "Properties > Properties > Miscellaneous > Toolbar > Elements", specify the buttons to be displayed in the object.



8. Under "Properties > Events" you can assign system functions or scripts to the buttons in the GRAPH overview in order to jump to the alarm control and the PLC code display in runtime and to open the TIA Portal.

Result

The GRAPH overview is inserted in the screen. The current status of the GRAPH step sequence is displayed in Runtime.

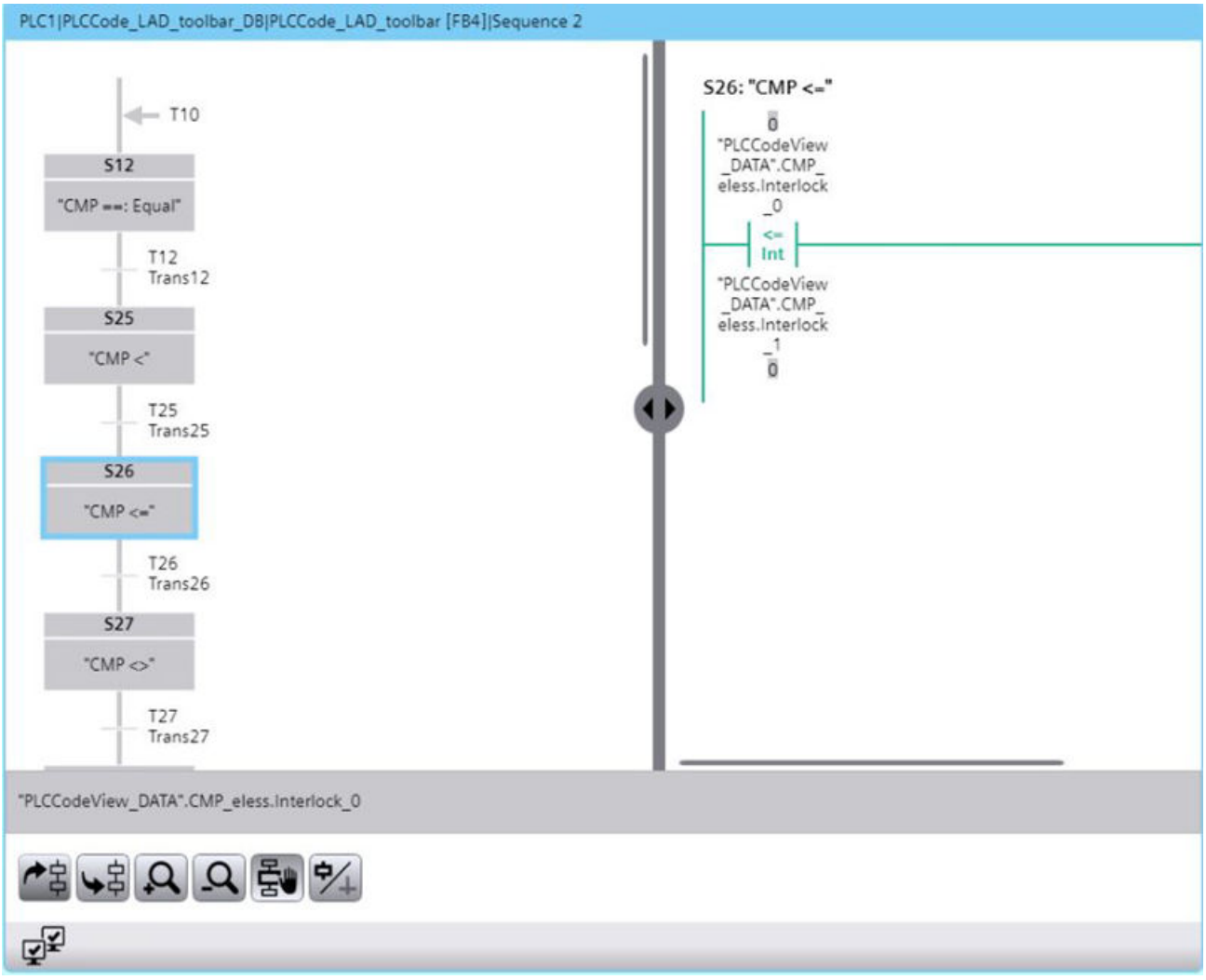
12.3.6 PLC code view

Use

The "PLC code viewer" object is used to display the current program status of user programs that have been programmed in the GRAPH programming language.

In the PLC code view, you display various items of information about the user program:

- Information area
- Toolbar
- Detail view
- Transition/Interlock view



Information area

The information area shows the GRAPH sequence in the left area and the details, e.g. for the step or for the transition, in the right area.

Toolbar

The toolbar shows information about the first or the selected icon.

Buttons of the toolbar

The table below shows the buttons on the toolbar and their meaning.

Operating element	Description	Function
	"Previous network"	Navigates to the previous network.
	"Next network"	Navigates to the next network.
	"Zoom in"	Enlarges the information area.
	"Zoom out"	Reduces the information area.
	"Step mode"	Switches between manual and automatic step selection for the active step.
	"Transition or Interlock"	Switches between the transition and interlock networks.

12.3.7 Configuring the PLC code view

Introduction

To display the PLC program networks in the GRAPH programming language in Runtime, insert a PLC code viewer control into your project.

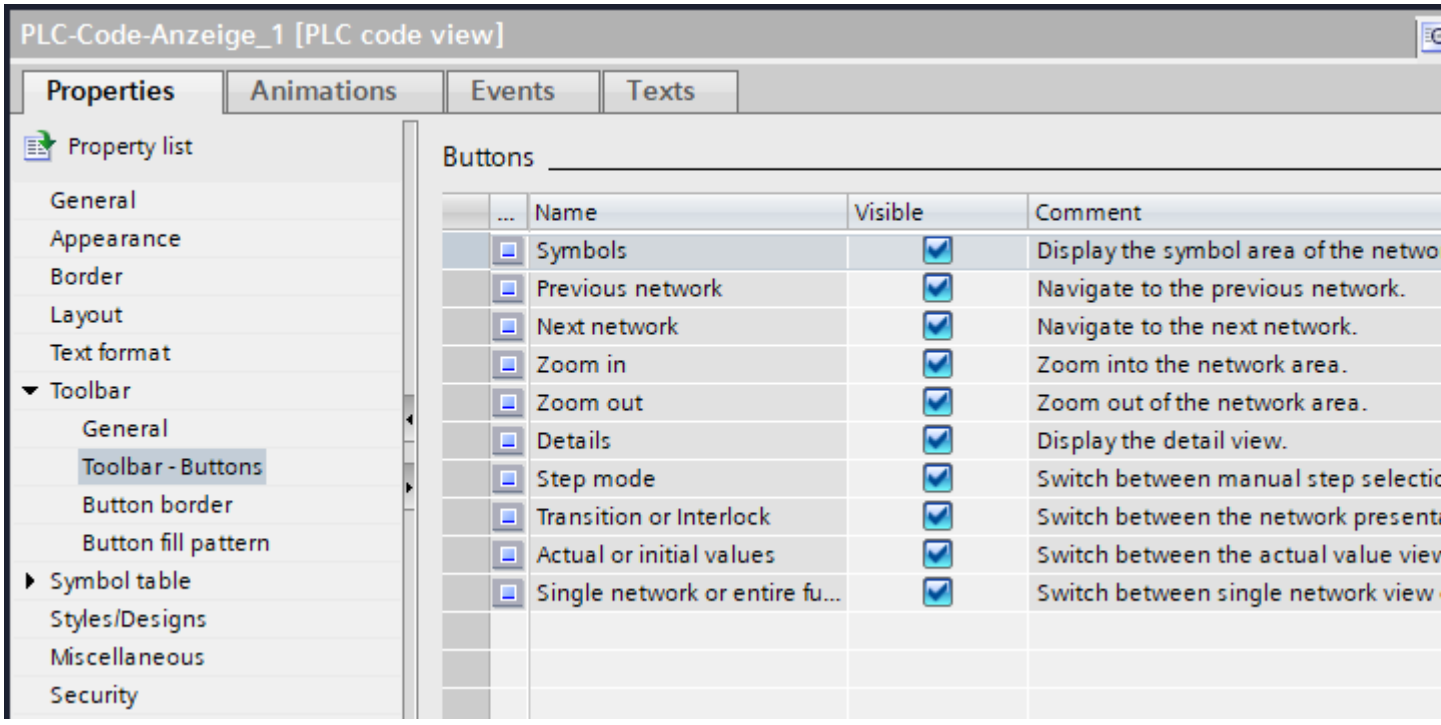
Requirement

- At least one PLC has been created.
- An HMI device has been created.
- An HMI connection has been established between the controller and HMI device.
- The system diagnostics is activated on all devices
- You have created a screen.

Procedure

1. Drag-and-drop the PLC code view from the toolbox view.
2. In the Inspector window, click "Properties > Properties > Toolbar".

3. Select the buttons that you require in Runtime, for example: Next network, Previous network, Step mode.



4. Select an authorization for operator input in "Properties > Properties > Security".

Result

The PLC code view is inserted in the screen. In Runtime, PLC user programs that are programmed in the GRAPH programming language can be displayed.

You can populate the PLC code viewer using system functions, e.g. jump from the GRAPH overview, or you can select the corresponding parameters directly.

Configuring users and roles 13

13.1 Basics

13.1.1 User management in the TIA Portal

The user management in the TIA Portal allows for the plant-wide, central user management including optional connection of Microsoft Active Directories. The user management forms the basis for an efficient and integrated management of personalized access rights in the plant. The safety risks are significantly reduced through this approach. Thanks to the specific assignment of roles and rights to individuals, maintenance is minimized while achieving a high degree of transparency.

Functions

The user management in the TIA Portal offers:

- Central, cross-project management of user groups and users in the system.
- Import of user groups and users from Microsoft Active Directory.
- Failsafe performance thanks to redundant design of a User Management Control domain (UMC domain).
- Load distribution thanks to multiple UMC stations in one UMC domain.

You can also find this information on the Internet in the "Centralized user management in TIA Portal" tile on the Software in TIA Portal (https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal/software/tia-portal-options.html) page.

You can configure the user management for:

- Engineering System
- Runtime

In the engineering system, you specify whether you are using a local or a central user management. By default, the use of the local user management is specified in the engineering system.

Protection in the engineering system

In the engineering system, you can protect the project from unintentional or unauthorized changes.

When you set up project protection in the engineering system, you will become the project administrator. Only authorized users can open and edit the protected project.

To protect the project, follow these steps:

1. Click "Security settings > Settings > Password policies" in the project tree.
2. You can define your own password policies.
3. Click on "Project protection > Protect this project".
4. Specify the login information for the project administrator.
5. The project administrator with the system-defined role "Engineering administrator" is created. The project is protected.

Project protection cannot be revoked.

Protection in Runtime

In Runtime, you can protect:

- Access to Runtime.
- Operation of objects from authorized use.

You can also use the user management without project protection. The system-defined engineering roles are not available in an unprotected project.

User management for the runtime

All information for user management in runtime is available in the following sections.

See also

Configuring user management in the engineering system for Runtime (Page 6894)

13.1.2 Central user management and UMC

The TIA User Management Component (UMC) option allows for the setup of a project-wide, central user management. You can also apply user groups and users from a Microsoft Active Directory.

From the TIA Portal, you can add the centrally defined user groups and users to the user management of the project. To add user groups and users from UMC, the corresponding rights are required in UMC.

Installing the User Management Component UMC

To use the central user management, install the "User Management Component UMC" software package. The UMC installation file and the UMC documentation in English is available on the TIA Portal installation data storage medium ("..\support", "...\Documents\Readme\English").

Install the UMC files on the PC on which you are managing the data of the central user management.

Note

We highly recommend that you read the UMC documentation completely before you start working with the user management, especially the sections on "Secure Application Data Support (SADS)". SADS is mandatory for working with the user management in the TIA Portal.

Licenses for central user management

The number of UMC domains is cumulative.

The license is free for up to 10 users.

The following licenses are available if you are managing more users:

Software/license	Article number
TIA Portal User Management Component (UMC) Rental license for 100 user accounts and 365 days	6ES7823-1UE30-0YA0
TIA Portal User Management Component (UMC) Rental license for 4000 user accounts and 365 days	6ES7823-1UE10-0YA0

- Certificates of license (https://mall.industry.siemens.com/mall/en/WW/Catalog/Search?searchTerm=user%20management%20component&tab=Product)
- Details about license concepts: Software and licenses (https://new.siemens.com/global/en/products/automation/topic-areas/simatic/licenses.html)
- Trial license (https://support.industry.siemens.com/cs/ww/en/view/109772992) for registered users in the Siemens Industry Online Support. The software is subject to export restrictions.

13.1.3 Local and central user management

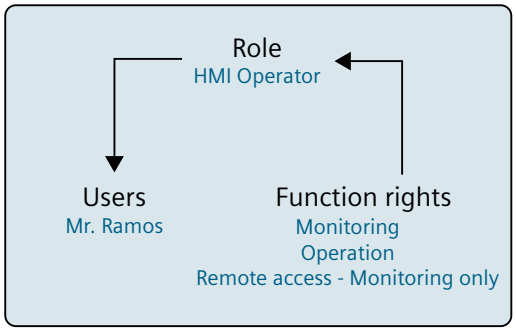
In the engineering system, you specify whether you are using a local or a central user management. By default, the use of the local user management is specified in the engineering system.

Local users and local user management

You define and manage the local users in a TIA Portal project. The local users are only valid for this project. You assign users different functions with system-defined or user-defined roles. You assign the function rights to the roles.

You cannot configure user groups in the local user management.

The following figure shows how you assign a role to a user and how you assign different function rights to the role:

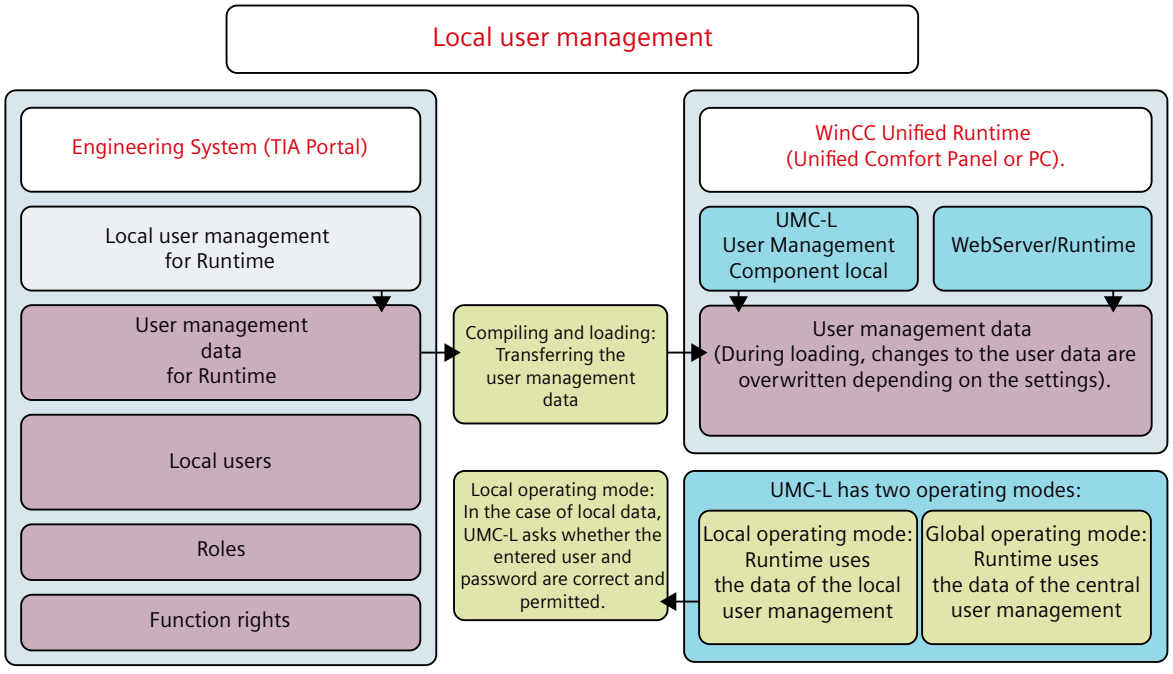


Options for local user management

A local user management is always part of the specific project in which the user management is configured.

The following figure shows the options of the local user management.

- Engineering system:
  - Configuring local users.
  - Adding roles and assigning function rights to the roles.
  - Assigning the roles to local users.
- WinCC Unified Runtime:
  - Managing local users.
  - Managing data of the local users.
  - Assigning the available roles to local users.



Logon via RFID with local user management for Unified PCs

If local user management is used, WinCC Unified Runtime supports login with RFID and PM-LOGON for local web clients.

The prerequisite is that PM-LOGON is installed on the UMC server and the teach-in of the RFID cards has been completed.

Additional information on licensing and installing PM-LOGON as well as on the teach-in for the RFID cards is available in the PM-LOGON Operating Manual (https://support.industry.siemens.com/cs/de/en/view/109810587).

Logon via RFID with local user management for Unified Control Panels

Additional information on using RFID on Unified Control Panels is available in the Unified Comfort Panels (https://support.industry.siemens.com/cs/ww/en/view/109810754) Operating Instructions TIA V18 or higher.

Central user groups, users and central user management

Outside of the TIA Portal, you define and manage the following in the central user management:

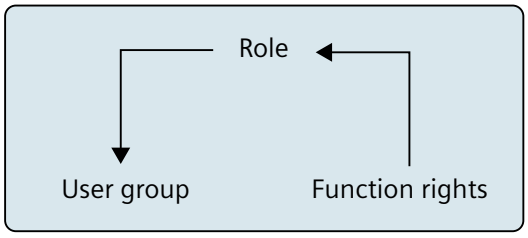
- User groups
- Central users

You organize the users in various user groups.

In the TIA Portal, you assign:

- The different functions to the user groups with the help of system-defined or user-defined roles.
- The function rights to the roles.

The following figure shows you how to you assign a role to a user group and function rights to the role:

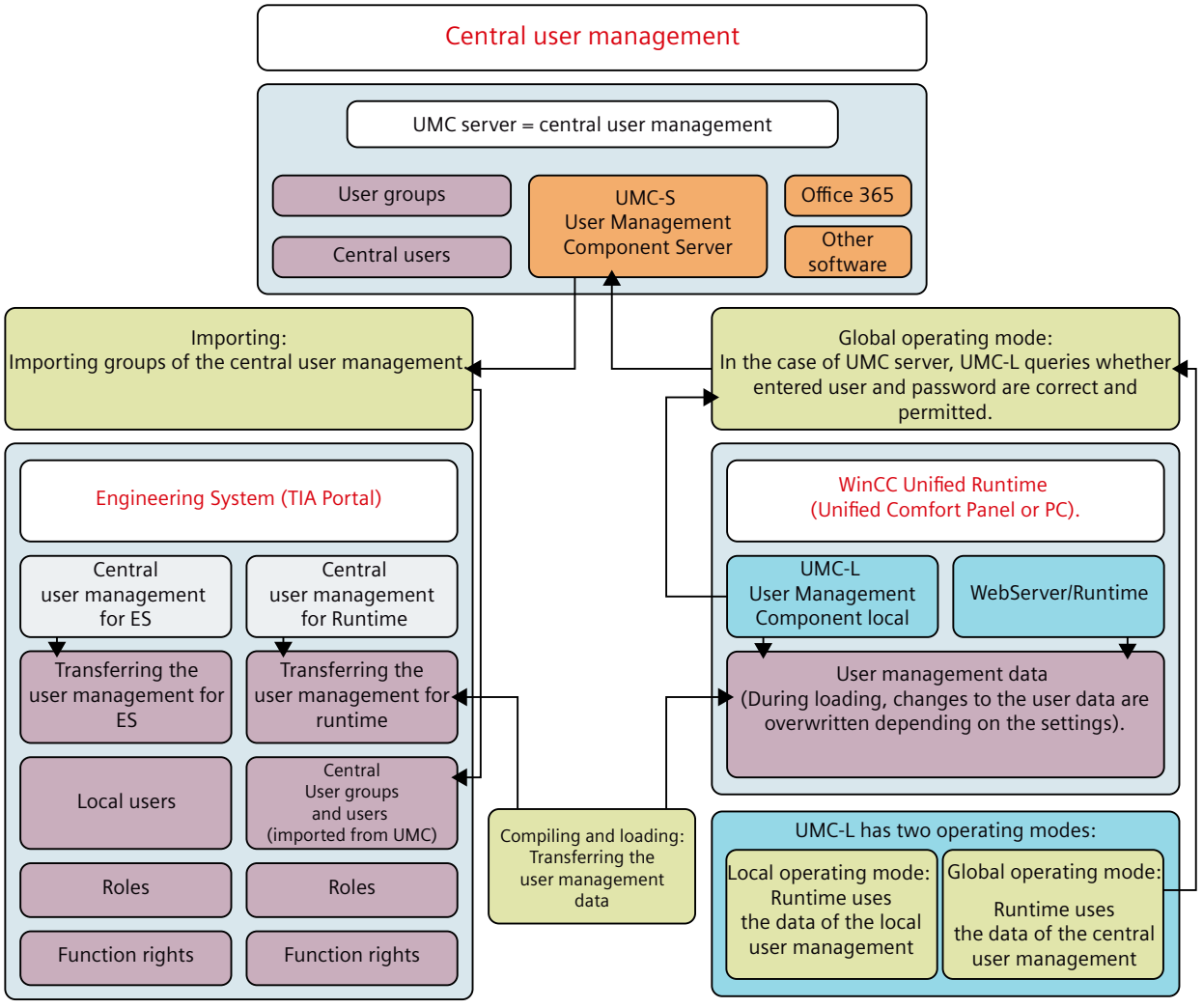


You can repeatedly import the user groups and central users from the central user management into the TIA Portal projects. You require the corresponding rights in the central user management for importing.

Options of the central user management

The following figure shows the options of the central user management.

- Engineering system:
  - Importing user groups and central users into a TIA Portal project.
- Central user management:
  - Managing user groups and their data.
  - Managing central users and their data.



See also

Examples (Page 6918)

13.1.4 Roles and function rights

In the user management, you assign users or user groups different functions through system-defined or user-defined roles. The roles are assigned system-defined and user-defined function rights by the system or by you.

- System-defined roles are specified by the system.
  You cannot rename or delete the system-defined roles. The function rights are assigned to the roles by the system. You cannot change the assignment.
- You can add user-defined roles.
  You can rename or delete the user-defined roles. You assign different function rights to a role.

Relevant roles in the engineering system

The system creates the following system-defined roles that are relevant for the engineering system.

Engineering roles:

- Engineering administrator
- Engineering standard

The roles are only available in a protected project.

Relevant roles in runtime

The system creates the following system-defined roles that are relevant for runtime.

HMI roles:

- HMI Administrator
- HMI Operator
- HMI Monitor
- HMI Monitor Client

The roles are also available in an unprotected project.

Function rights

The function rights specify which functions a user or a user group may use in a specific role. You can link different function rights to multiple roles. You can also add user-specific runtime rights.

You can assign multiple function rights to a role.

Note

User - Role - Function right

If you do not assign a role to a user or a function right to a role, the user or role is not downloaded to the device.

13.2 Configuring user management in the engineering system for Runtime

13.2.1 Specifying local or central user management

In the engineering system, you specify whether you are using local or central user management. By default, the use of the local user management is specified in the engineering system.

Requirement

- A project is open.
- A device has been created.
- Password settings for Runtime and engineering are defined.

Specify password settings

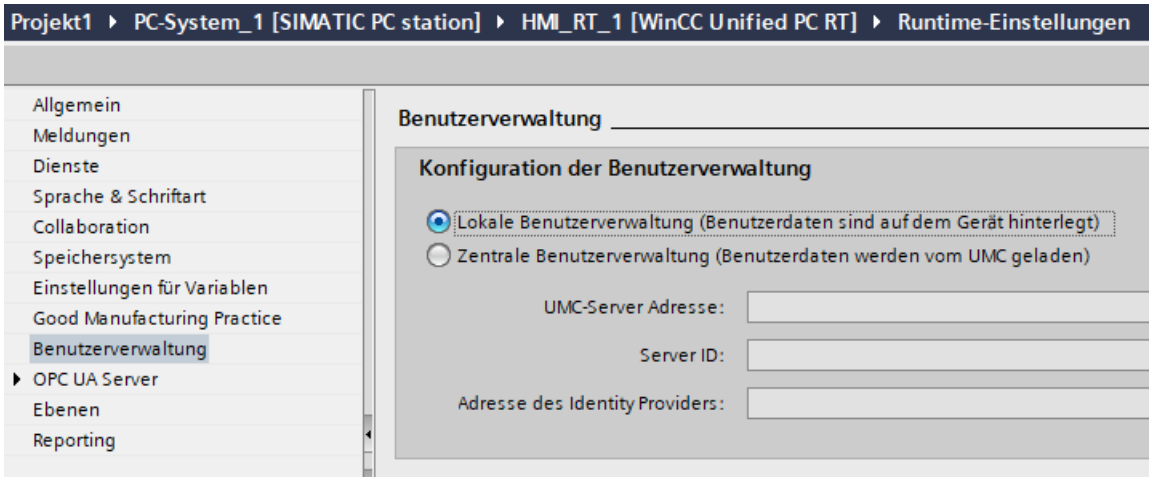
To set the password settings for the runtime and engineering, follow these steps:

1. Open "Security settings > Settings" in the project tree.
2. In the "Settings" editor, select the "Password policies" menu command.
3. Specify the password settings.

Specifying the user management

To define the user management, follow these steps:

1. Open the "Runtime settings" of the device in the project tree of the project.
2. Under "User management > User management configuration", select:
   - Local user management
     User data is stored on the device.
   - Central user management
     User data is loaded from the UMC.



See also

Managing local users (Page 6900)

Managing central users and user groups (Page 6903)

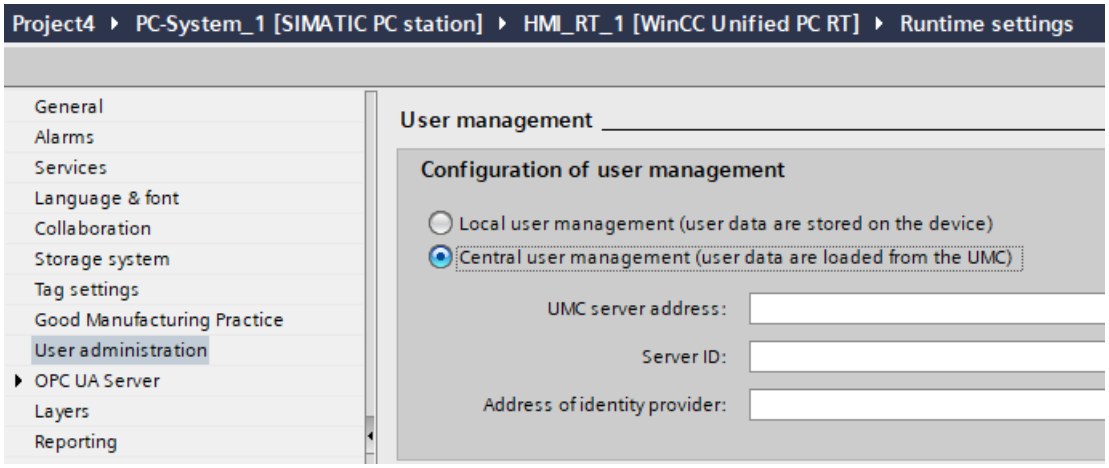
13.2.2 Configuring a connection to the central user management

You have specified in the Engineering System that you are using central user management.

Configuring a connection to the central user management

Follow these steps to configure the connection to the central user management:

1. Enter the UMC server address in "User management configuration".
2. Enter the server ID. The server ID can be determined from the configuration of the UMC server.
   The server ID is the fingerprint of the web certificate (https) on which the UMC server is installed. For more information, refer to Server ID (Page 6896).
3. Enter the address of the identity provider.



You are connected with the central user management.

Note

If you cannot connect to the central user management or you do not know the server ID, leave the entry field for the server ID empty.

The configuration is nevertheless downloaded to Runtime. Enter the server ID:

1. On the PC using Runtime Manager.
2. On the Panel using the Control Panel.

13.2.3 Server ID

The WinCC Unified PC station or the Unified Comfort Panel communicate via an encrypted HTTPS connection to the UMC server to verify the authentication.

The server ID is the fingerprint of the Internet Information Service (IIS) web certificate (HTTPS) on the computer where the UMC server is installed.

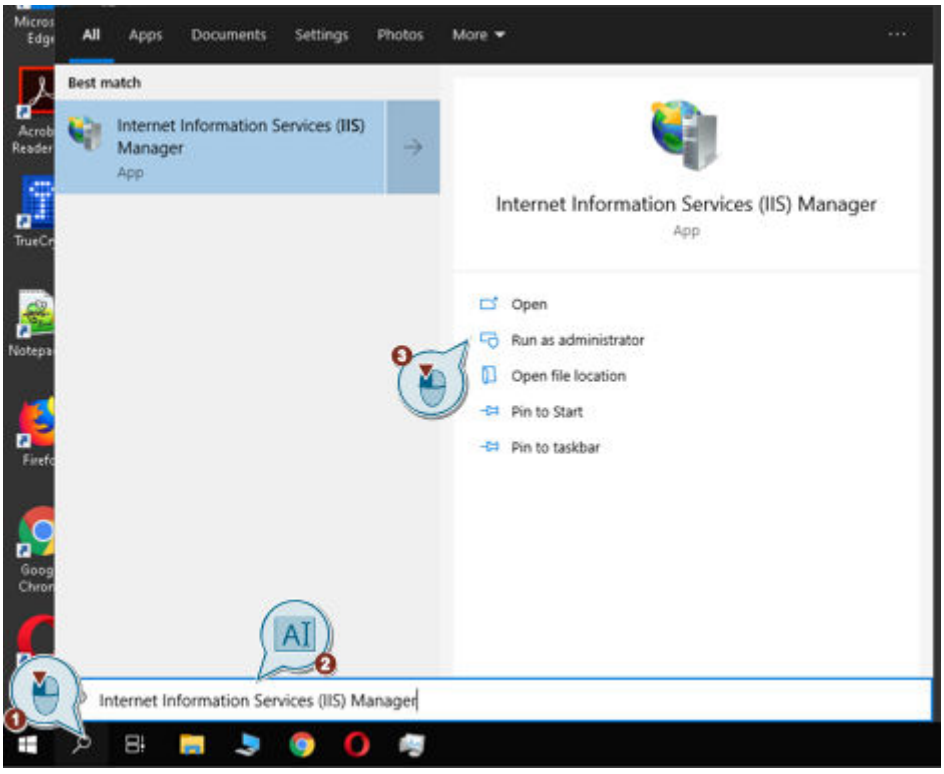
Note

In a distributed UMC domain with UMC ring server and multiple UMC servers / UMC RT servers, you decide which server of the UMC domain is used for user authentication of the respective Unified Station.

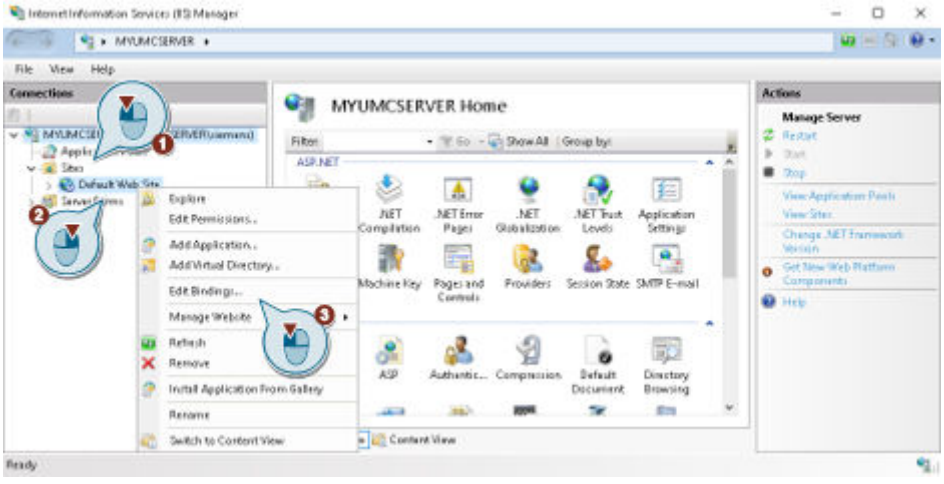
Take over server ID via the IIS Manager

If you want to apply the server ID via the IIS Manager, follow these steps:

1. Open the IIS Manager on the computer to which the WinCC Unified PC station or the SIMATIC WinCC Unified Comfort Panel is to connect for authentication.



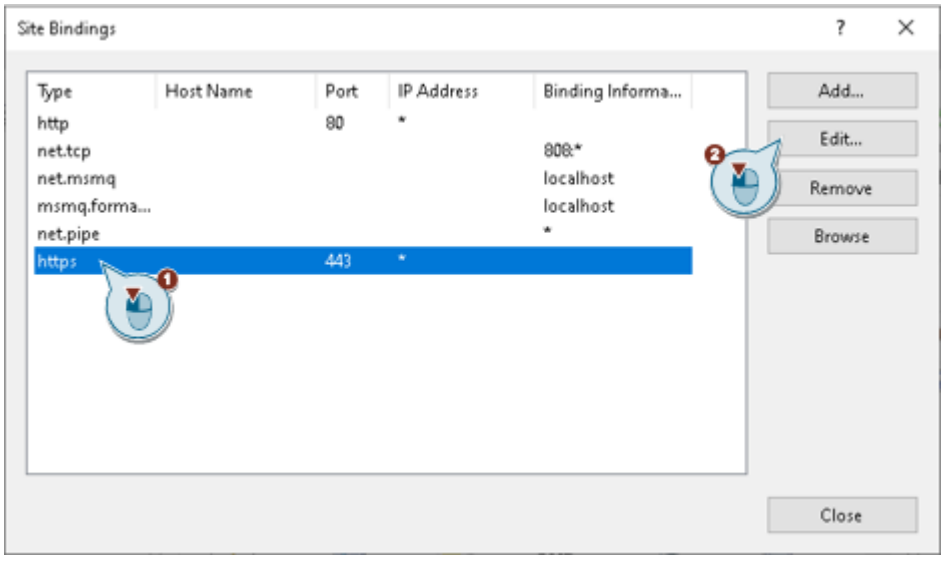
2. Edit the bindings of the default web page in IIS.



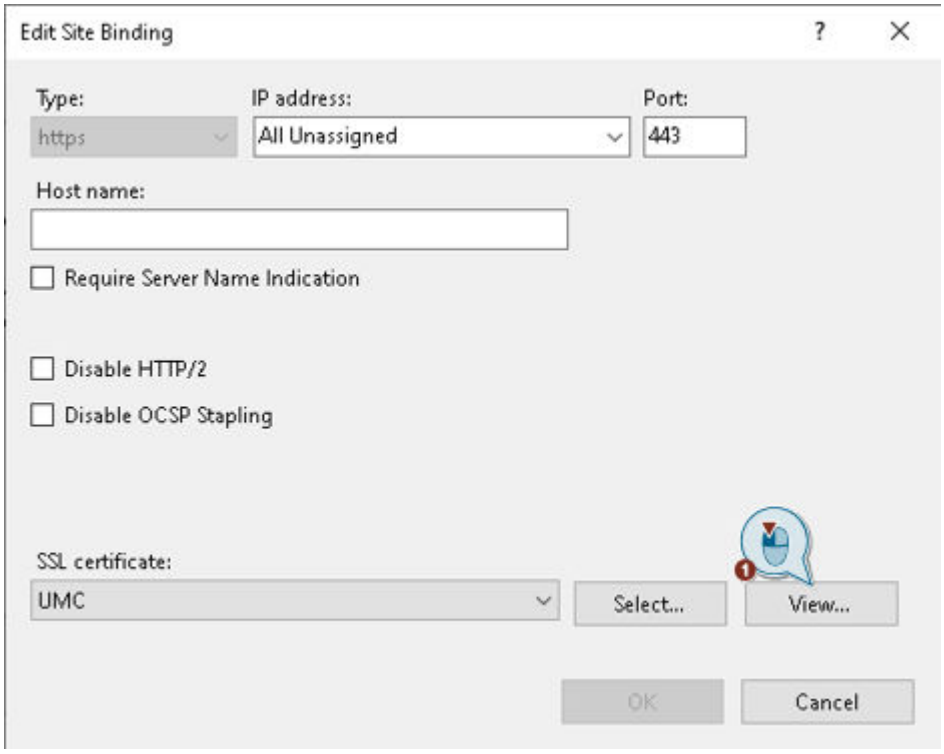
Note

If WinCC Unified is already installed on the same computer (single-user station system), select the "WinCC Unified SCADA" web page as the web page.

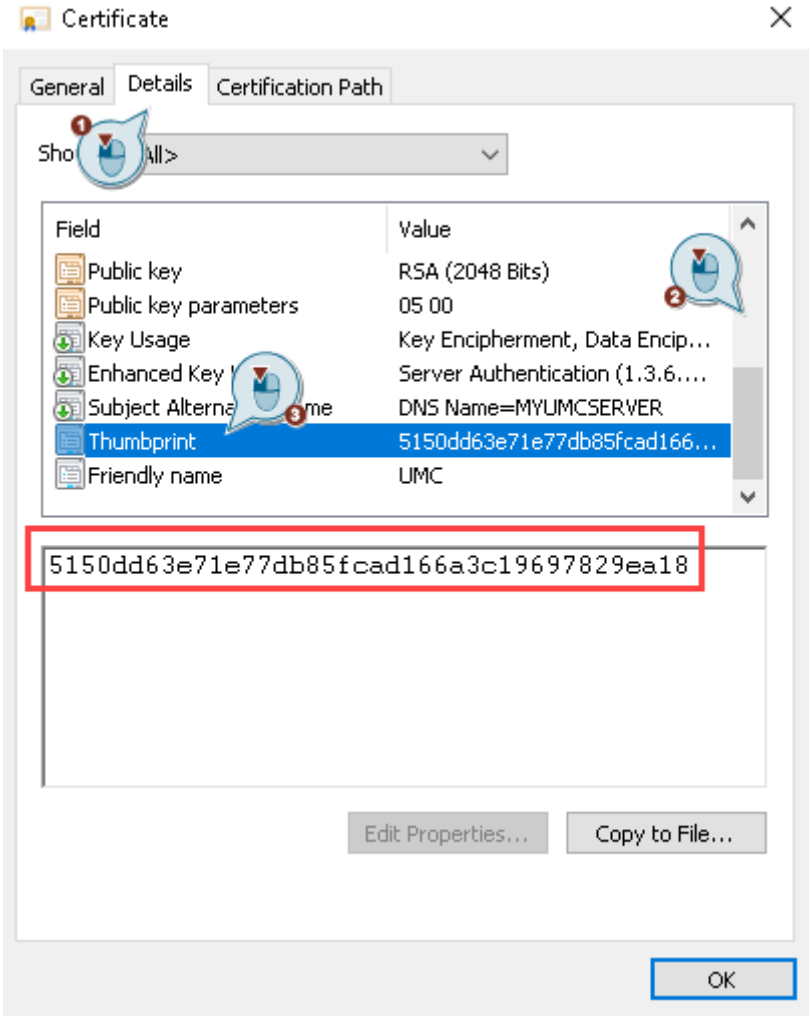
3. Edit the bindings of the 'HTTPS' type.



4. Open the stored SSL certificate.



5. Open the details of the SSL certificate. Scroll to the "Fingerprint" attribute.



6. Transfer the fingerprint of the certificate into the runtime settings of the WinCC Unified Station.

Take over server ID via a browser

You can also take over the server ID via a browser and without administrator rights.

You can find the procedure under the following link to "Fingerprints (https://www.grc.com/fingerprints.htm)" in the section "How to display this page's (or any page's) SSL certificate fingerprint".

13.2.4 Users and user groups

13.2.4.1 Managing local users

You have specified in the Engineering System that you are using local user management. To manage the local users in your project, you have the option of:

- Adding users
- Specifying data of a user:
  - User name
  - Password
  - Authentication procedure
  - Runtime timeout
  - Comment
- Copying, pasting, deleting users

Requirement

- A project is open.
- The "Security Settings > Users and Roles" editor is open.
- The "Users" tab is open.

Restrictions

Please note the following restrictions:

- You can add a maximum of 256 users.
- The user name may not exceed 255 characters.
- The password may not exceed 120 characters and must meet the defined password guidelines.
- The comment may not exceed 1000 characters.

Note

Permissible characters for user names and passwords

You can use the following numbers, letters and special characters for user names and passwords:

- 0123456789
- A...Z a...z
- !#$%&()*+,-./:;<=>?@\[]^_`{}~|
- Spaces within the user name or password

Adding users

To add a user, follow these steps:

1. Click "Add new user".
2. Click "Add new local user" in the submenu.
3. Enter a user name according to the password policies.
4. Select the authentication procedure for the user:
   - Password: Enter a password according to the password policies. Enter the password again to confirm.
   - Radius: The logon is made via a RADIUS server on which the password is saved. This option is only used for devices that support logon using a RADIUS server.
5. Set the runtime timeout for the user. This setting is only effective on the Unified Comfort Panel. Observe the notes in the individual manuals.
6. You can enter a comment for the project user.

A new project user has been created. Assign the roles to the user.

	**Tips for an efficient procedure**
• You can also create a new user by copying an existing user. As a result, the assigned roles are also assigned to the copied user. You must re-assign the password for the copied user.	

"Anonymous" user

In V18, a default "Anonymous" user is available. When enabled, the "Anonymous" user does not require a password.

You cannot use the "Anonymous" user to log in to Runtime.

Changing the user data

To change the data of a user, follow these steps:

1. In the "Users" tab, click the field whose data you want to change.
2. Change the user name, password, authentication procedure, runtime timeout or comment.

Deleting users

To delete a project user, proceed as follows:

1. Select the user in the "Users" tab that is open.
2. In the shortcut menu, select the "Delete" command or press the <Del> key.

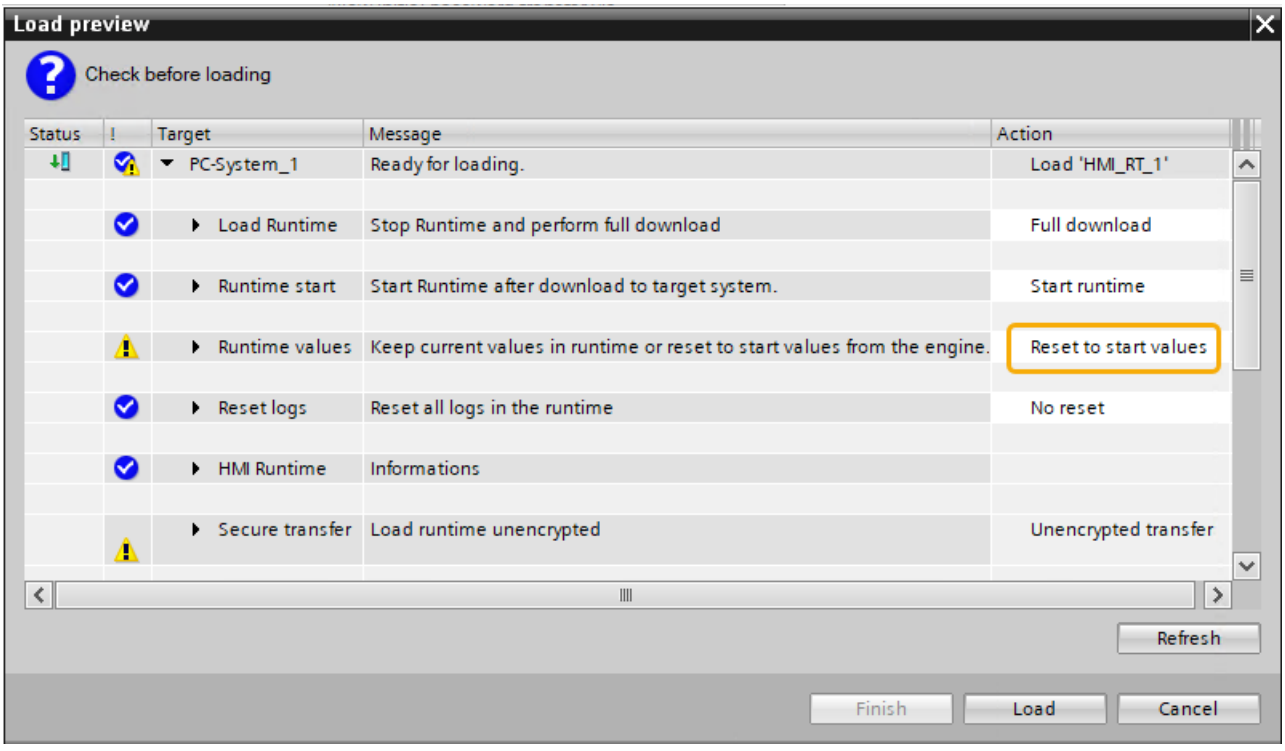
13.2.4.2 Downloading local user management

You have added at least one user. Compile and download the local user management.

Compiling and downloading local user management

To load the local user management for the first time, follow these steps:

1. Select the device in the project tree.
2. Select "Download to device > Software (all)" from the shortcut menu.
   The compilation of the project is checked and content that has not been compiled is compiled. The compilation result is displayed in the Inspector window under "Info > Compile".
3. The "Load Preview" dialog is displayed.
4. Check the displayed defaults and change the settings if necessary:
   - Specify whether runtime should start on the target system after the download.
   - Before the download, specify whether existing data is retained. When downloading for the first time, select "Reset to initial values".
   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".



5. Click "Download".

Runtime starts after loading the settings.

Note

For the first download of a TIA Portal project to a runtime device, select the option "Reset to start values" when downloading the user data.

For all subsequent loading processes, select "Keep current values".

On the runtime device, the user configuration is not project-specific, but device-specific. The defined user management is used in all projects in a device.

All system-defined roles are included in each runtime project.

Keep current values during download

To keep the current values during the next download, select "Keep current values" during download.

The following user management data is kept:

- User name
- Password
- Password guidelines
- Comment
- Language
- Function rights
- User role
- User group
- User management mode - local or central
- Data of the central user management: Server address, server ID, identity provider address

See also

Managing local users (Page 6900)

13.2.4.3 Managing central users and user groups

You have specified in the Engineering System that you are using central user management.

You can add central users and user groups that were created in the central user management to a project. Central users and user groups are managed in the central user management. The changes affect the project to which these users or user groups belong.

You can synchronize the user management in the engineering system with the central user management or check the synchronization status.

Main advantage of the central user management

The main advantage of the central user management is the management of the user groups.

- In the user groups, you manage the central users independent of the engineering system. You add the central users to the user groups.
- The function rights of a user are known via the role assignment in the user group in the Unified Runtime.
- Changes to the user data are effective in Runtime even without loading the project.
- When you import the user group into the TIA Portal, the users and their data including passwords is automatically imported into the project.

Requirement

- A project is open.
- The connection to the central user management is configured.
- You have a user account with the corresponding rights in the central user management.
- The "Security Settings > Users and Roles" editor is open.
- The "User groups" tab is open.

Restrictions

Please note the following restrictions:

- You can add a maximum of 256 central users.
- You can add a maximum of 50 central user groups.
- You cannot copy central users and user groups.

Information about user groups and users

You can display the following information for central users and user groups:

- User groups: The users they contain and whether they have already been imported into the user management.
- User: The user groups of which the user is a member and whether they have already been imported into the user management.

Adding user group

To add a user group from the central user management to a TIA Portal project, follow these steps:

1. Open the "User groups" tab.
2. Click "Add new user group".
   If you have not yet logged in to the central user management yet, the "UMC Login" dialog opens:
   - Enter your UMC user name and the corresponding password.
   - Click "OK".
3. If you are logged in to the central user management, the "Add new user group from UMC" dialog opens. All available user groups from the central user management are displayed. Already added user groups are activated and write-protected.
4. Activate the user groups that you want to add to the TIA Portal project. Click "OK".
5. The selected user groups are added as central user groups. Their data is write-protected, and you cannot change the data within the TIA Portal project.

Deleting a user group

To delete a user group from the TIA Portal project, follow these steps:

1. Open the "User groups" tab.
2. Select one or more user groups.
3. In the shortcut menu, select the "Delete" command or press the <Del> key.

Adding central users

You can also import the individual central users into a TIA Portal project.

To add a central user from the central user management, follow these steps:

1. Open the "Users" tab.
2. Click "Add new user".
3. Click "Add global users" in the submenu.
   If you have not yet logged in to the central user management yet, the "UMC Login" dialog opens:
   - Enter your UMC user name and the corresponding password.
   - Click "OK".

   If you are logged in to the central user management, the "Add new user from UMC" dialog opens. All available users from the central user management are displayed. Already activated users are activated and write-protected.
4. Activate the user that you want to add to the TIA Portal project. Click "OK".
5. The selected users are added as central users. Their data is write-protected, and you cannot change the data within the TIA Portal project.

	Tips for effective procedure
•	To find the required users, filter the table by the columns "Name" and "Long name". Enter the name in the first line. All users with this name are displayed. To cancel filtering, click on the "Filter" button or select "*" in the drop-down list.

Deleting a central user

To delete a central user from the TIA Portal project, follow these steps:

1. Open the "Users" tab.
2. Select one or more central users.
3. In the shortcut menu, select the "Delete" command or press the <Del> key.

See also

Server ID (Page 6896)

13.2.4.4 Loading central user management

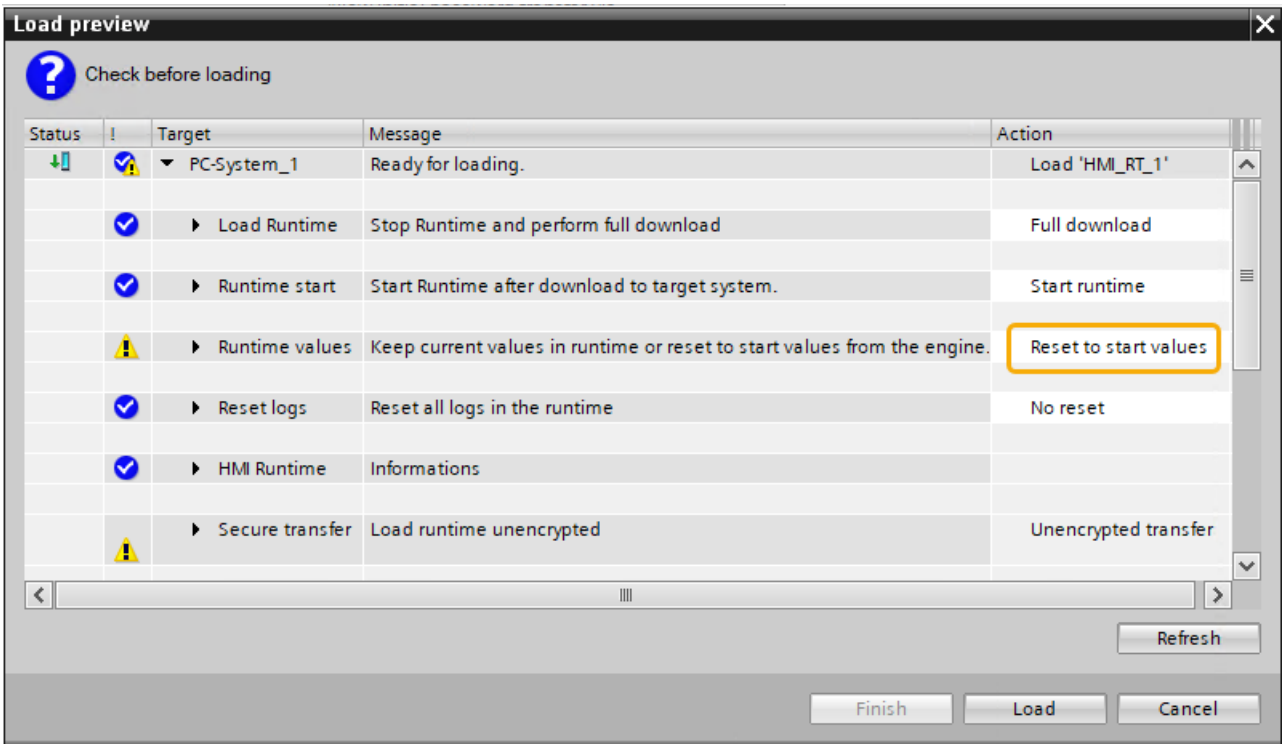
You have added at least one user group. Compile and load the central user management.

Compiling and loading central user management

To download all data of the central user management, follow these steps:

1. Select the device in the project tree.
2. Select "Download to device > Software (all)" from the shortcut menu.
   The compilation of the project is checked and content that has not been compiled is compiled. The compilation result is displayed in the Inspector window under "Info > Compile".
3. The "Load Preview" dialog is displayed.

4. Check the displayed defaults and change the settings if necessary:
   - Specify whether runtime should start on the target system after the download.
   - Before the download, specify whether existing data is retained. When downloading for the first time, select "Reset to start values".
   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".



5. Click "Download".

The connection to the central user management is established after loading the settings.

Depending on the selected options, runtime is started during loading.

Note

For the first download of a TIA Portal project to a runtime device, select the option "Reset to start values" when downloading the user data.

For all subsequent loading processes, select "Keep current values".

On the runtime device, the user configuration is not project-specific, but device-specific. The defined user management is used in all projects in a device.

All system-defined roles are included in each runtime project.

Note

Enter the settings loaded for the central user management:

- On the PC in the SIMATIC Runtime Manager
- On the Panel in the Control Panel.

Depending on the setting, data is overwritten during loading.

Keep current values during download

If you want to keep the current values during the next download, select "Keep current values" during download.

The following user management data is kept:

- User name
- Password
- Password guidelines
- Comment
- Language
- Function rights
- User role
- User group
- User management mode - local or central
- Data of the central user management: Server address, server ID, identity provider address

Behavior of the runtime after the download

If all your entries are correct, runtime uses the downloaded settings.

If the entries are incomplete or incorrect, runtime exhibits the following behavior:

- If the data of the central user management has already been downloaded successfully once before, runtime uses the user data that was downloaded last.
  Enter missing settings for central user management:
  - On the PC in the SIMATIC Runtime Manager
  - On the Panel in the Control Panel.
- If the data of the central user management has never been downloaded, runtime does not start.
  Enter missing settings for central user management:
  - On the PC in the SIMATIC Runtime Manager
  - On the Panel in the Control Panel.

13.2.5 HMI roles

13.2.5.1 Managing HMI roles

You assign users the roles with the function rights. You can manage the user-defined HMI roles in your project.

System-defined roles relevant for HMI

System-defined HMI roles without engineering function rights are created in an unprotected project:

- HMI Administrator
- HMI Operator
- HMI Monitor
- HMI Monitor Client

You cannot rename or delete system-defined HMI roles. Also, you cannot change the assignment of the function rights to system-defined roles.

User-defined roles relevant for HMI

You can perform the following actions to manage the user-defined HMI roles in your project:

- Add a new user-defined role.
- Change the data of the user-defined role:
  - Name
  - Runtime timeout
  - Comment
- Assign the function rights to the role.
- Change or delete the assignment of the function rights.
- Delete a role.

Requirement

- A project is open.
- A device has been created.
- The "Security Settings > Users and Roles" editor is open.
- The "Roles" tab is open.

Adding a user-defined HMI role

To add a user-defined HMI role, follow these steps:

1. Double-click "Add new role".
2. Enter a name, runtime timeout and a comment, if required.

A user-defined HMI role has been created. Assign the function rights to the role.

	**Tips for effective procedure**
• You can also create a user-defined HMI role by copying an existing HMI role. As a result, the assigned function rights are also assigned to the copied HMI role.	

Change the data of a user-defined role

To change the data of a user-defined role, follow these steps:

1. Click in the field whose data you want to change.
2. Change the role name, runtime timeout or comment.

Note

When setting the runtime timeout, observe the notes in the individual manuals.

Delete user-defined role

To delete a user-defined role, follow these steps:

1. Select the user-defined role.
2. In the shortcut menu, select the "Delete" command or press the <Del> key.

13.2.5.2 Assigning HMI roles

You can assign HMI roles with different function rights to local users and global user groups.

Requirement

- A project is open.
- A device has been created.
- Users and user groups have been created.
- The "Security Settings > Users and Roles" editor is open.

Assigning HMI roles to local users

To assign roles to a user, follow these steps:

1. Open the "Users" tab.
2. Select the user.
3. Activate the desired HMI roles in the "Assigned roles" tab.

Assigning HMI roles to global user groups

To assign roles to a global user group, follow these steps:

1. Open the "User groups" tab.
2. Select the user group.
3. Activate the desired HMI roles in the "Assigned roles" tab.

Revoking role assignments for local users

To revoke a role for a user, follow these steps:

1. Open the "Users" tab.
2. Select the user.
3. In the "Assigned roles" tab, disable the roles that you no longer want to assign to the user.

Revoking role assignments for global user groups

To revoke a role assignment for a global user group, follow these steps:

1. Open the "User groups" tab.
2. Select the user group. Note that you cannot use multiple selection.
3. In the "Assigned roles" tab, clear the roles that you no longer want to assign to the user group.

13.2.5.3 HMI role "HMI Monitor Client"

The system-defined role "HMI Monitor Client" contains the function right "WinCC Unified Client Monitor - Limited Access". No other function rights can be assigned to this role.

The "HMI Monitor Client" role allows you to monitor and analyze the production process without influencing the processes in the PLC unintentionally or without authorization.

The following description applies to Unified PC and Unified Comfort Panel.

Note

To operate the "HMI Monitor Client", you need one or more "HMI Monitor Client" licenses depending on the number of accesses.

Important notes

The "HMI Monitor Client" HMI role is superior to all other roles and their function rights. A user who is assigned the "HMI Monitor Client" role is only granted the functional rights of this role. Function rights with a higher precedence, of other roles that are assigned to the user, are lost.

Restrictions of the "HMI Monitor Client" role

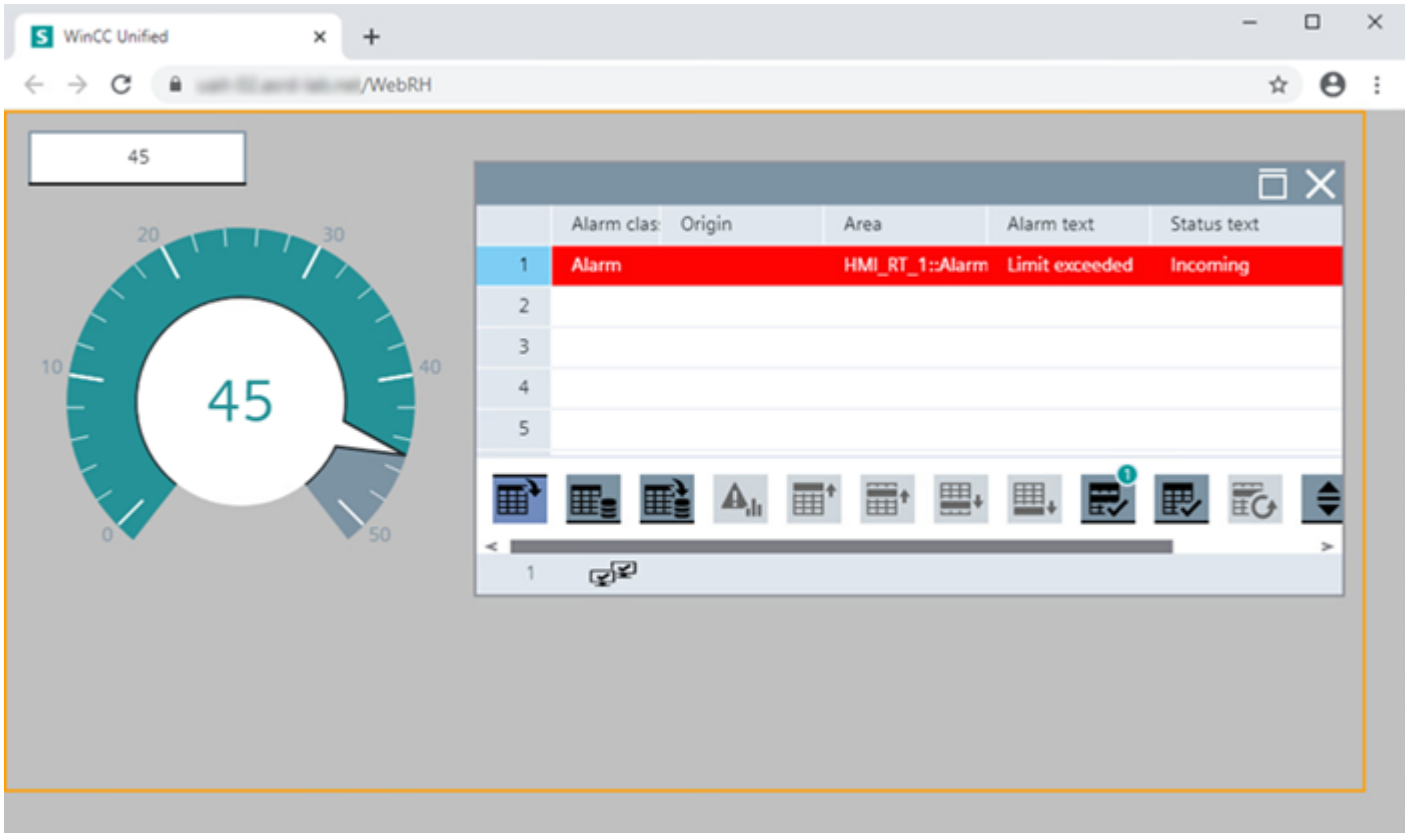
A user with the function right "WinCC Unified Client Monitor - Limited access" may monitor the processes but only operate them to a limited extent.

The following table shows which operations the user cannot carry out:

Tags	Transfer value changes from external tags (PowerTags) to the PLC
Data logging	Clear logs
	Write log values manually
	Comment log values
	Write correction values
Alarms	Acknowledge active alarms
	Reset active alarms
	Disable alarms
	Enable alarms
Alarm logging	Delete alarm log
	Comment logged alarms
Parameter set control	Process values cannot be written to the PLC with the "Write to PLC" button of the parameter set control.
	Process values cannot be written to the PLC with the system function "LoadAndWriteParameterSet".
	Process values cannot be written to the PLC with the snippet "Load Parameter Set from storage and write to PLC".
	Parameters sets cannot be loaded from the parameter set memory using control tags. Process values cannot be written to the PLC using control tags.

Visualization in Runtime

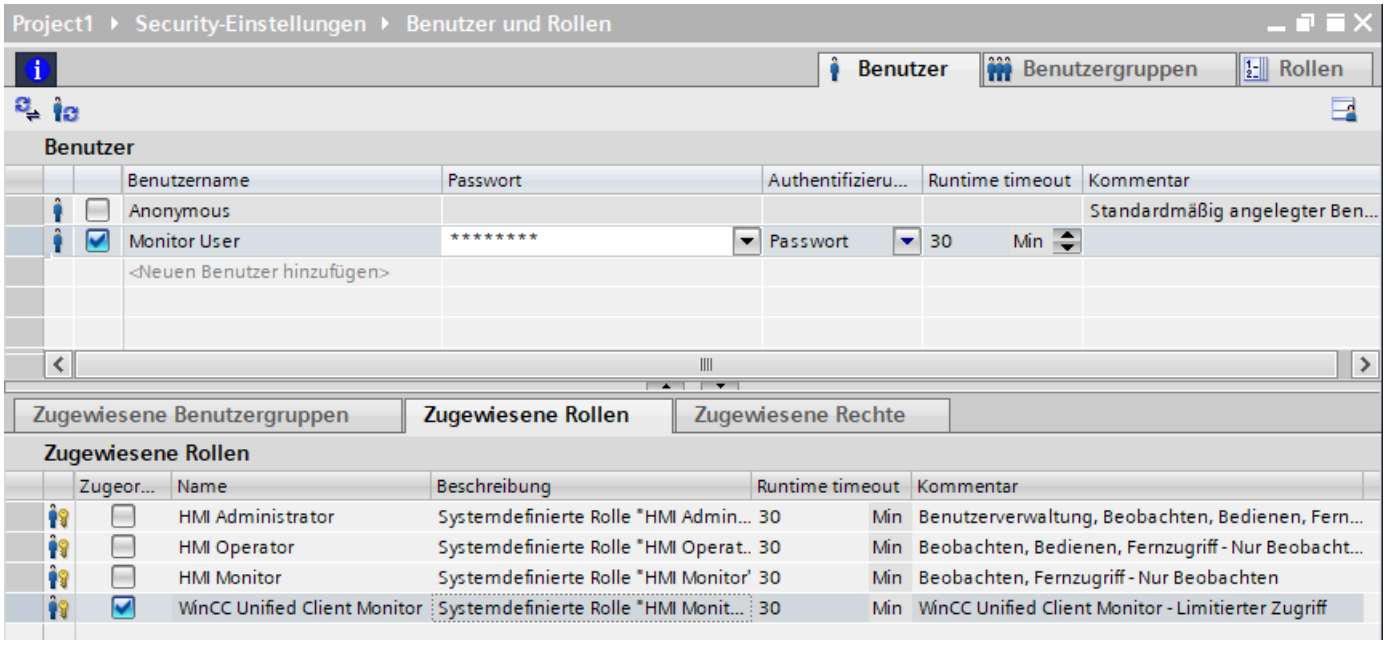
In Runtime, the Monitor Client mode is visualized by an orange frame around the root screen.



Activating Monitor Client mode

To activate the Monitor Client mode, follow these steps:

1. Create a user or select an existing user.
2. Assign one of the following roles to the user:
   - The system-defined role "HMI Monitor Client".
   - A user-defined role that contains the function right "WinCC Unified Client Monitor - Limited access".



3. The user logs on to Runtime.
4. The Monitor Client mode is activated.

The user is allowed to monitor the process but not to change any values that are relevant for the process. These values are listed at the top of the table.

See also

System-defined function rights (Page 6914)

Managing HMI roles (Page 6909)

13.2.6 Function rights

13.2.6.1 System-defined function rights

You can assign the different function rights to the HMI roles. These function rights specify which functions a user may use in Runtime.

System-defined function rights

The following table shows an overview of the system-defined function rights in the TIA Portal. The function rights are automatically checked and activate the associated functionality.

You cannot rename or delete the system-defined runtime rights.

Function right	Description	HMI roles
Importing and exporting users	The user may import and export user management data in runtime. The function right is not supported on a Unified Comfort Panel.	User-defined role
User management	The user has permission to manage the existing users and add and manage the new users in Runtime.	HMI Administrator
Enable HMI Monitor Client	The user may observe the process on an HMI device to a limited extent without unintentionally or unauthorizedly influencing the processes in the PLC.	HMI Monitor Client
OPC UA read and write access	The user may access the data of a different device with read and write permission via OPC UA.	HMI Administrator
Openness Runtime read and write access	The user may access the data of a Unified runtime with read and write permission via Openness.	HMI Administrator
Reset UMC password	The user may reset the UMC password.	User-defined role
Control Panel access	The user may change the settings in the Control Panel of a Unified Comfort Panel.	HMI Administrator

Additional function rights may be available that are associated with additional products and options that are installed.

Function rights to the screen objects

The following table shows an overview of the function rights that are set on the screen objects. These function rights have no pre-defined meaning.

The meaning of the function rights comes from the objects that are protected and trigger an action.

Function right	Example application	HMI roles
Operate	The user may operate access-protected objects in runtime.	HMI Administrator HMI Operator
Monitor	The user may only monitor but not operate the objects.	HMI Administrator HMI Operator HMI Monitor
Remote access	The user may use Unified Collaboration to access an HMI device.	HMI Administrator
Remote access - Monitor only	The user may use Unified Collaboration to access an HMI device. The user may only monitor but not operate the objects.	HMI Administrator HMI Operator HMI Monitor

See also

Protecting the Control Panel from being accessed (Page 6926)

HMI role "HMI Monitor Client" (Page 6911)

13.2.6.2 User-defined function rights

You can assign different runtime rights to the HMI roles. These function rights specify which functions a user may use in WinCC Unified Runtime in Runtime.

You can implement your protection concepts with the user-defined runtime rights.

Restrictions

Please note the following restrictions:

- You can add a maximum of 999 user-defined Runtime rights.
- The name of the user-defined runtime right and the name of the group of runtime rights can be up to 128 characters long.
- The comment may not exceed 1000 characters.

Note

You can use the following numbers, letters, and special characters for the names and groups of the user-defined runtime rights:

- 0123456789
- A...Z a...z
- !#$%&()*+,-./:;<=>?@\\[]^_`{}~|
- Spaces within the user name or password

Requirement

- A project is open.
- A device has been created.
- The "Security Settings > Users and Roles" editor is open.
- The "Roles" tab is open.

Adding a user-defined Runtime right

To add a user-defined runtime right, follow these steps:

1. Click on the "User defined runtime rights" tab.
2. In the "Function rights" area, double-click "Add new right" in the "Name" column.
3. Enter a name, a group and a comment, if required.

Note

The "Group" column is used to group the user-defined runtime rights in the engineering system. The assignment of the runtime rights to a group has no effect on the runtime.

A user-defined runtime right has been created. Assign the runtime right to a role.

	Tips for effective procedure
•	You can also create a user-defined runtime right by copying an existing runtime right.

Changing the data of a user-defined runtime right

To change the data of a user-defined runtime right, follow these steps:

1. Click in the field whose data you want to change.
2. Change the name, the group or the comment.

Deleting a user-defined runtime right

To delete a user-defined runtime right, follow these steps:

1. Select the user-defined runtime right.
2. In the shortcut menu, select the "Delete" command or press the <Del> key.

13.2.6.3 Assigning function rights to an HMI role

You assign the function rights to the roles.

Assign function rights to a user-defined role

To assign the function rights to a user-defined role, follow these steps:

1. Select the user-defined role.
2. In the "Runtime rights" tab, open the category from which you want to assign the function rights.
3. In the "Function rights" area, enable the function rights that you want to assign to the role.

Changing or deleting the assignment of the function rights

To change or delete the assignment of the function rights to a user-defined role, follow these steps:

1. Select the user-defined role.
2. In the "Runtime rights" tab, open the category from which you want to assign the function rights or revoke the assignment:
   - In the "Function rights" area, enable the function rights that you want to assign to the role.
   - In the "Function rights" area, disable the function rights that you no longer want to assign to the role.

Displaying assigned function rights of a local user

To display the assigned function rights of a local user, follow these steps:

1. Open the "Users" tab.
2. Select the user.
3. Open the "Assigned rights" tab in the lower area.
4. Expand the categories of function rights in the "Categories of function rights" column.
   The assigned function rights are displayed in the "List of rights" column.
   The roles by which the function rights are assigned to the user are displayed in the "Rights derived from role" column.

Display assigned function rights of a global user group

To display the assigned function rights of a global user group, follow these steps:

1. Open the "User groups" tab.
2. Select the global user group.
3. The assigned function rights are displayed in the lower area of the "Assigned rights" tab.

13.2.7 Examples

13.2.7.1 Example: Setup of the local user management

Task

In the following example, you set up a user management for different users. The example orientates itself to a typical requirement profile from manufacturing engineering.

Principle

Users with various roles are involved in a project. Create the users and assign them to the roles.

You can reproduce different views through the roles.

Example:

- Organizational view: Commissioners, Operators, Shift I, Shift II.
- Technological view: Axis control, Tool changers, Plant North, Plant South.

The following example orientates itself to the organizational view.

Each user has function rights for specific applications.

In the example, you create Mr. Meier, Ms. Ramos, Ms. Greenwood, Messrs. Peters and Santini. You assign roles to each user.

Requirements

- A new project has been created.
- The "Users and roles" editor in "Security settings" is open.

Procedures overview

To assign the users one or more roles, follow these steps:

1. Create the users.
2. Create the roles and assign the users one or more roles. Note the following information:
   - The function rights have been defined for the system-defined roles.
   - You assign the required function rights to the user-defined roles.
3. Configure a button with access protection.

Result

The goal is the following structure of the user management consisting of users, roles and function rights:

The user "Meier" has the "HMI administrator" role.

Note

Not all function rights that have been specified for the system-defined "HMI Administrator" role are displayed in the table.

Users	Role	Function rights		
		User management	Operation	Remote access
Meier	HMI Administrator	x	x	x

You assign the required roles to the other persons.

Users	Role	Function rights							
		Operation	Monitoring	Remote access	User-defined runtime right Screen change	User-defined runtime right Toggle language	User-defined runtime right Change parameter sets	User-defined runtime right Delete parameter sets	User-defined runtime right Stop Runtime
Ramos	HMI Operator	x	x	x					
Greenwood	Operator				x	x			
Peters	Line-Manager				x	x	x	x	
Santini	Service				x	x	x	x	x

13.2.7.2 Example: Add user and assign to a role

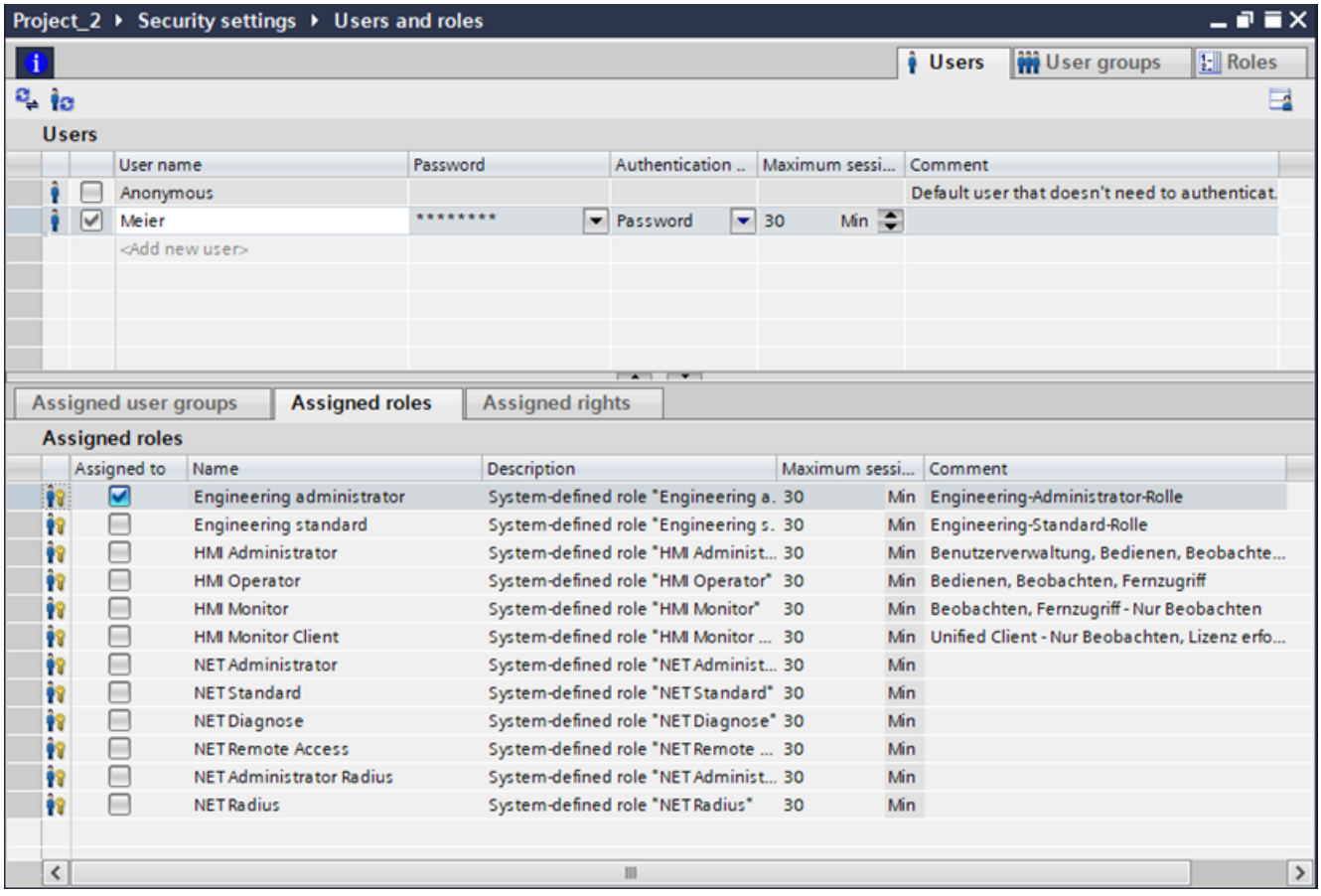
Task

In the following example, you create the users and assign them to their roles. The users are sorted alphabetically immediately after the name has been entered.

Procedure

1. Open the "Users" work area.
2. Double-click "Add new user" in the "Users" table.
3. Enter "Meier" as the user name.
4. Click the  button in the "Password" column. The dialog box for entering the password is opened.
5. Enter "Meier123" as the password. The password must meet the defined password policies.
6. To confirm the password, enter it a second time in the lower field.
7. Close the dialog box by using the  icon.
8. Enable the "HMI Administrator" role in the "Assigned Roles" table.

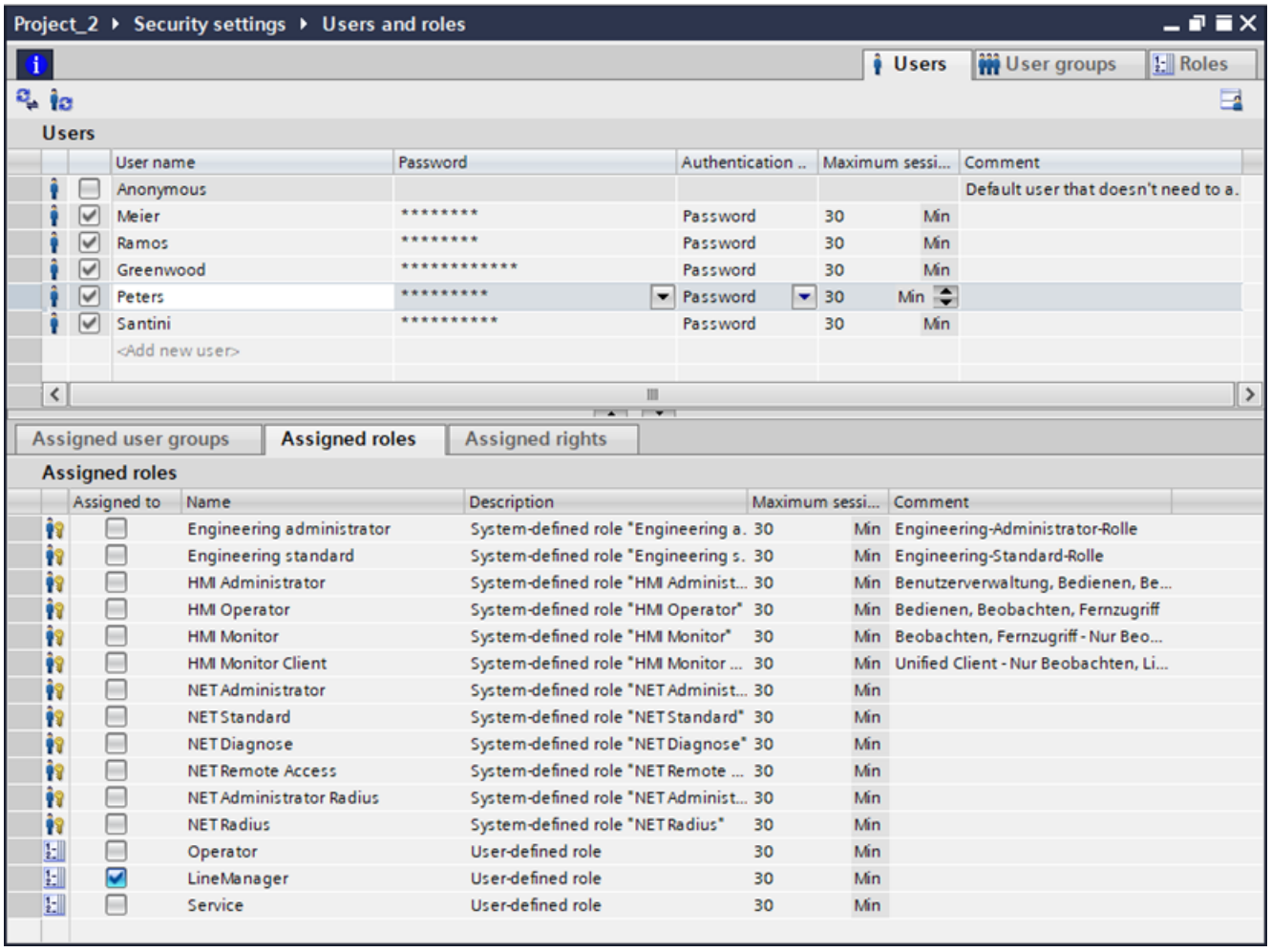
Interim result



Procedure

1. Double-click "Add new user" in the "Users" table.
2. Enter "Ramos" as the user name.
3. Click the button in the "Password" column. The dialog box for entering the password is opened.
4. Enter "Ramos123" as the password.
5. To confirm the password, enter it a second time in the lower field.
6. Close the dialog box by using the icon.
7. Enable the "HMI Operator" role in the "Assigned Roles" table.
8. Repeat steps 2 to 6 for the users "Greenwood", "Peters" and "Santini".
9. Enable the user-defined roles "Operator", "LineManager" and "Service" of the individual users in the "Assigned Roles" table.

Result



13.2.7.3 Example: Add roles and assign function rights

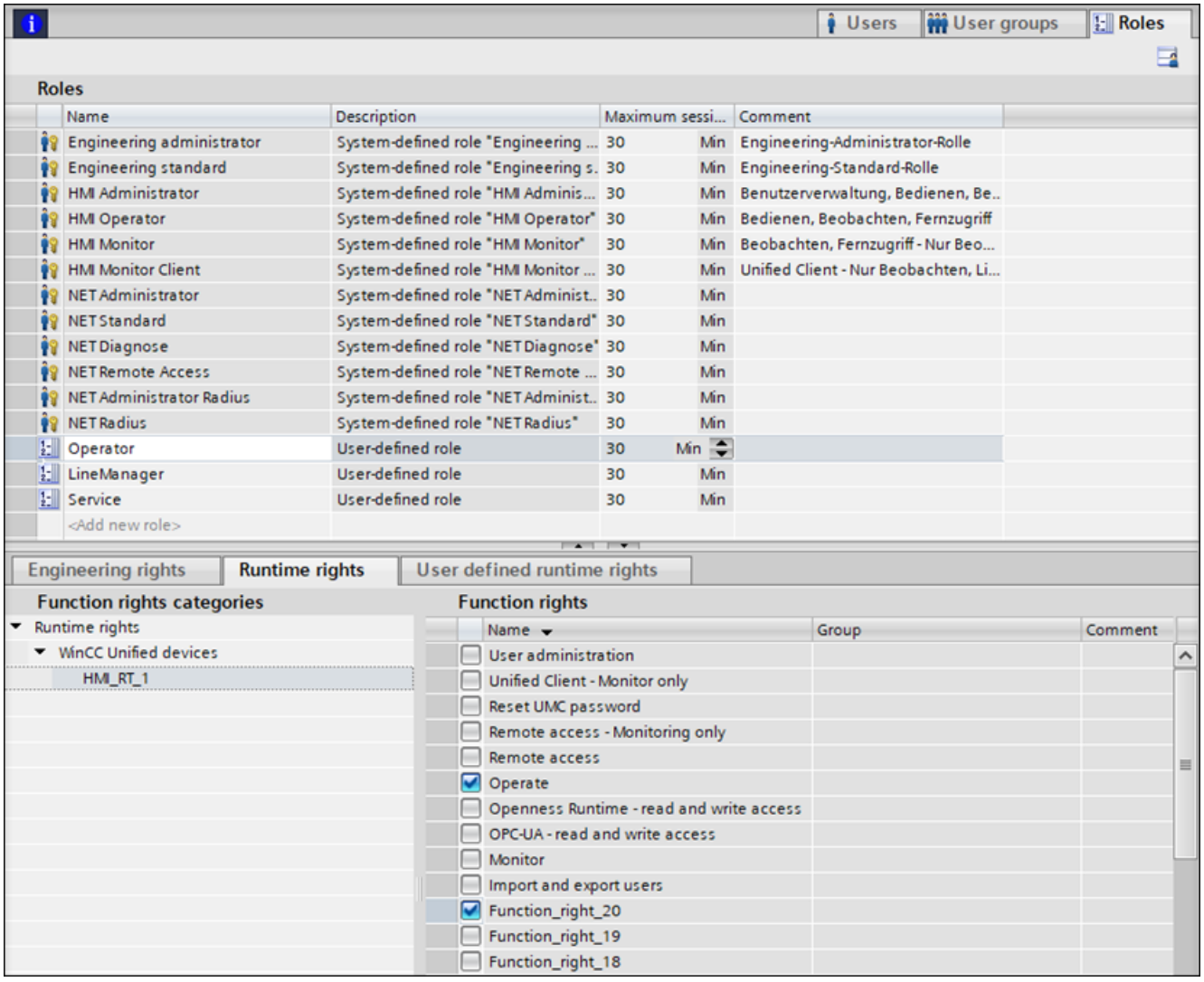
Task

In the following example, you create the roles and assign the function rights to the roles.

Procedure

1. Open the "Roles" work area.
2. Double-click "Add new role" in the "Roles" table and enter a name for the role.
3. In the "Runtime rights" table, click on "Runtime rights > WinCC Unified devices > HMI_RT_1" in the "Categories of function rights" column.
4. In the "Function rights" column, assign the desired function rights to the role.

Result



13.2.7.4 Example: Configuring a button with access protection

Task

In the following example, you use a system function to create a button for a screen change. You protect the "To Recipe view" button against unauthorized operation. To do so, you configure the "Change parameter sets" function right at the "To Recipe view" button.

Requirements

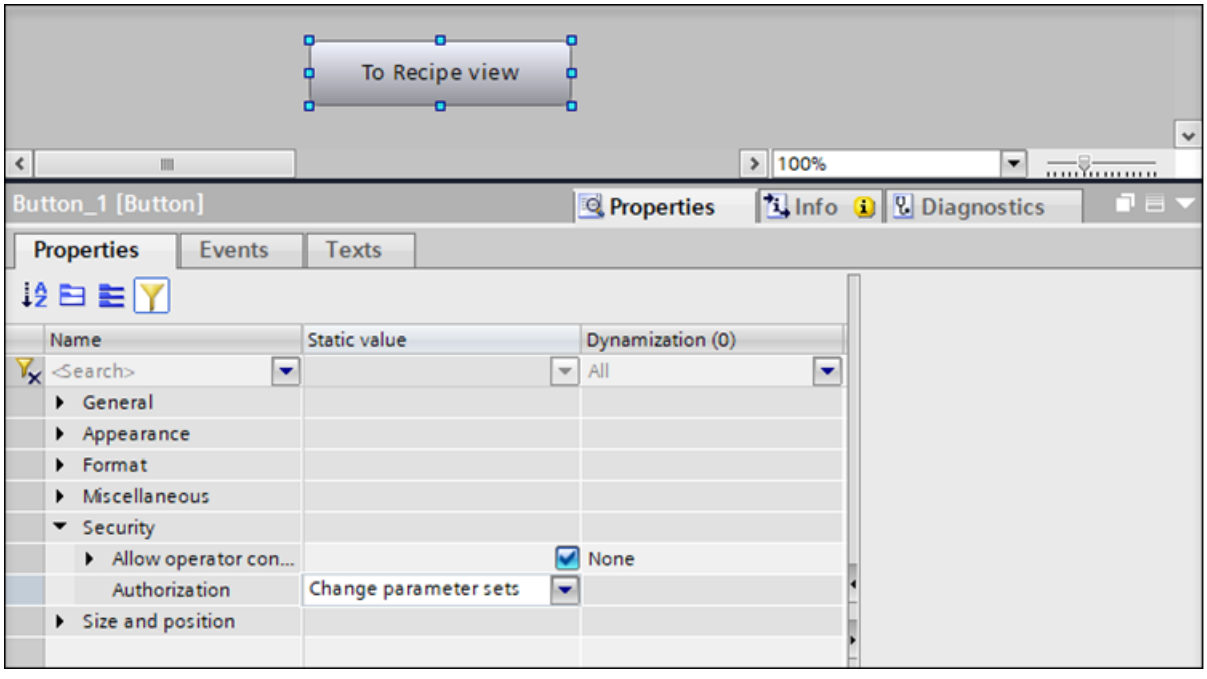
- A "Change parameter sets" function right has been created.
- A "Recipes" screen has been created.

- A "Start" screen has been created and opened.
- A button has been created and marked in the "Start" screen.

Procedure

1. In the Inspector window, click "Properties > Properties > General > Text".
2. Enter "To Recipe view" as the text.
3. Click "Properties > Events > Click left mouse button" in the Inspector window.
4. Click the "Add function" entry in the first line of the "Function list" table.
5. Select the system function "ChangeScreen" from the "Screen" group.
6. Click on the ... button in the "Screen name" row of the "Value" column. A dialog box for selecting the screen opens.
7. Select the "Recipes" screen and use the button to close the dialog box.
8. Click "Properties > Properties > Security" in the Inspector window.
9. Select "Change parameter sets" as function right.

Result



Access to the "To Recipe view" button is protected. If the user "Greenwood" clicks the button in Runtime, for example, the "Recipes" screen opens. Requirement:

- The user "Greenwood" has logged on correctly and has the required function right.
- The "Recipes" screen contains a recipe view and other screen objects.

Note

If the logged-on user does not have the required function right or if no user is logged on, the "Logon dialog box" is displayed. An alarm indicating that no operator authorization is available for this user appears in Runtime.

13.3 Using the user management on the Unified Comfort Panel

13.3.1 Notes on commissioning

Commissioning with central user management

To ensure that the connection to the central user management is set up during initial commissioning of the Panel, we recommend that you do not activate any access protection for the Control Panel on the Panel.

If the access protection is not activated for the Control Panel, configure the connection to the central user management in the Control Panel.

If the access protection for the Control Panel is activated and no connection could be set up during commissioning for the central user management, you have the following options:

- Download the project and the settings for user management from the engineering system.
- Reset the Panel to the factory settings.

Note

Failure of the network connection

If the network connection to the central user management fails, all users who have already logged on to the project once before can still log on for some time. When the connection to the central user management has been restored, other users can log on once again as well.

13.3.2 User management on the Unified Comfort Panel

In the engineering system you specify whether you want to work with local or central users and user groups from the central user management on a Unified Comfort Panel. By default, the use of the local user management is specified in the engineering system.

Note

You can only switch between local and central user management in the engineering system.

To manage users in Runtime, you require the "User management" function right. Configure a user with the required rights in the engineering system and load the user into Runtime.

13.3.3 Protecting the Control Panel from being accessed

In the engineering system you can protect the Control Panel of a Unified Comfort Panel from unauthorized access. You can assign a user the access right for changes on the Control Panel.

Requirement

- A project is open.
- A user has been created in the engineering system.
- A user-defined role has been created.
- A Unified Comfort Panel has been created.

Assign access right to Control Panel to a user

To assign a user the function right "Control Panel access" to access the Control Panel, follow these steps:

1. Open the "Security settings" folder in the project tree.
2. Double-click on "Users and roles".
   The "Users and roles" editor opens in the work area.
3. Open the "Roles" tab.
4. Select a user-defined role.
5. In the lower area "Function rights categories", open the category of the Runtime rights.
6. Click on the category of the Unified Comfort Panels.
7. In the lower area "Function rights", activate the function right "Control Panel access ".
8. Open the "Users" tab.
9. Select a user.
10. In the area "Assigned roles", activate the user-defined role to which you have assigned the function right for access to the Control Panel.

The user receives the access right for the Unified Control Panel and is permitted to change the Panel settings.

Activate access control for Control Panel

In addition, activate the access control in the Control Panel:

1. Start the Panel.
2. Open the Control Panel.
3. Under "Security > Control Panel Access", activate the access control for the Control Panel.

Note

In contrast to the Unified PC Runtime, no login in required. When you open the protected Control Panel, your user rights are checked. If you do not have the necessary rights, a login window is displayed.

See also

System-defined function rights (Page 6914)

Using central user management in the Control Panel (Page 6940)

13.3.4 Managing local users

13.3.4.1 Options for local user management

To manage local users in Runtime, the user requires the "User management" function right on the HMI device.

Managing local users

You have the following options for managing local users:

1. Control Panel of the Unified Comfort Panel. You can find all information in the "SIMATIC HMI devices Unified Comfort Panels (https://support.industry.siemens.com/cs/ww/en/view/109773257)" manual.
2. Via the "Browser" screen object in a project.
3. Using an Internet browser on a PC.

13.3.4.2 Using local user management in the Control Panel

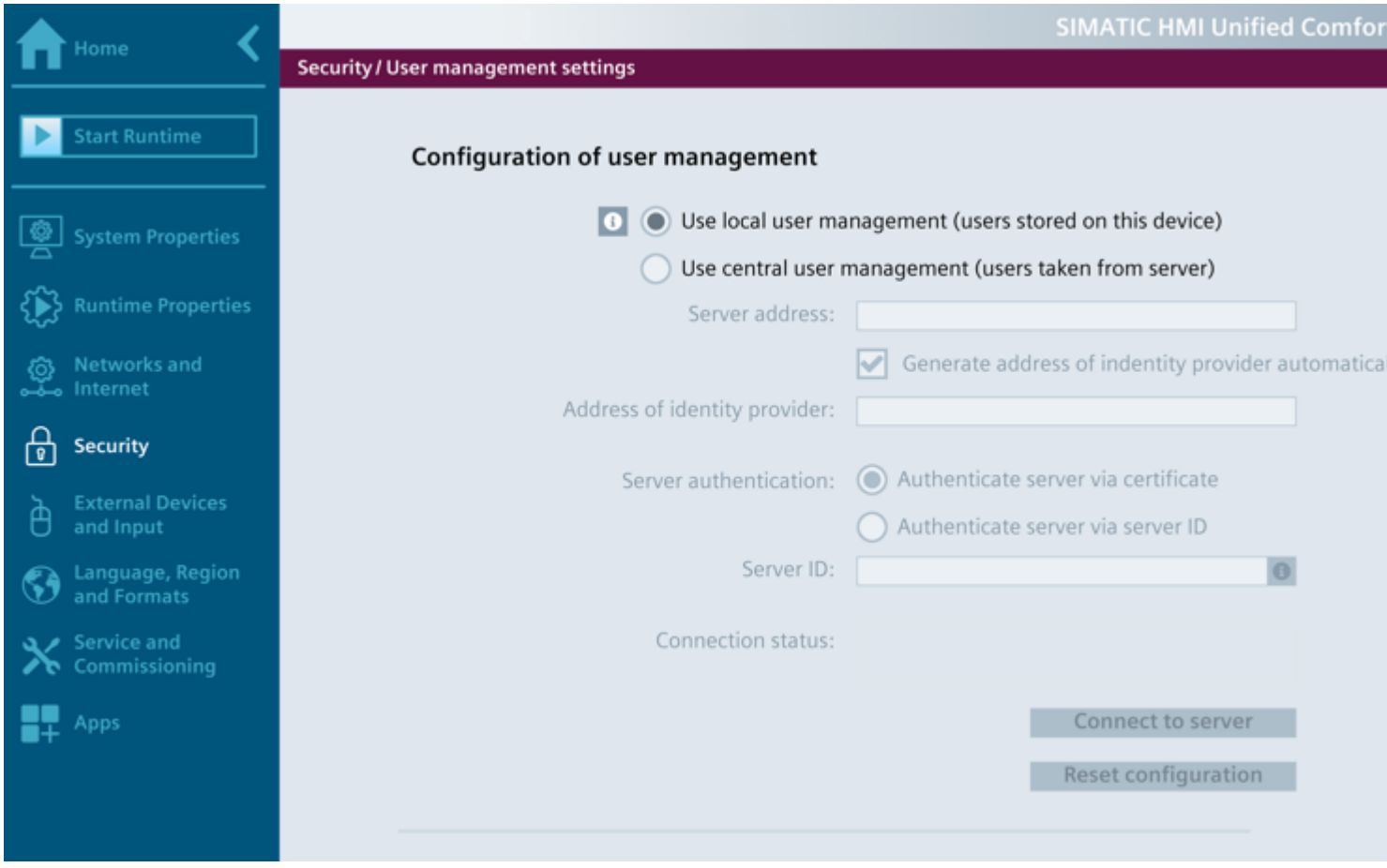
If access protection is configured for the Control Panel in the engineering system, you need the "Control Panel access" function right. If you do not have the necessary rights, a login window is displayed.

You cannot make any changes to the user management in the Control Panel.

Local user management in the Control Panel

If you want to use the user management on the Unified Comfort Panel, follow these steps:

1. Select "Security" in the navigation.
2. Click "User management settings".
3. In the "Configuration of user management" dialog, you can see whether the local or central user management is enabled.



13.3.4.3 Opening local user management in the "Browser" screen object

In Runtime, you access local user management of a Panel by using the "Browser" screen object in a screen.

The detailed description of the individual steps in Runtime can be found in the section "Managing local users in Runtime (Page 6930)".

Note

The specific possible operations depend on the function right.

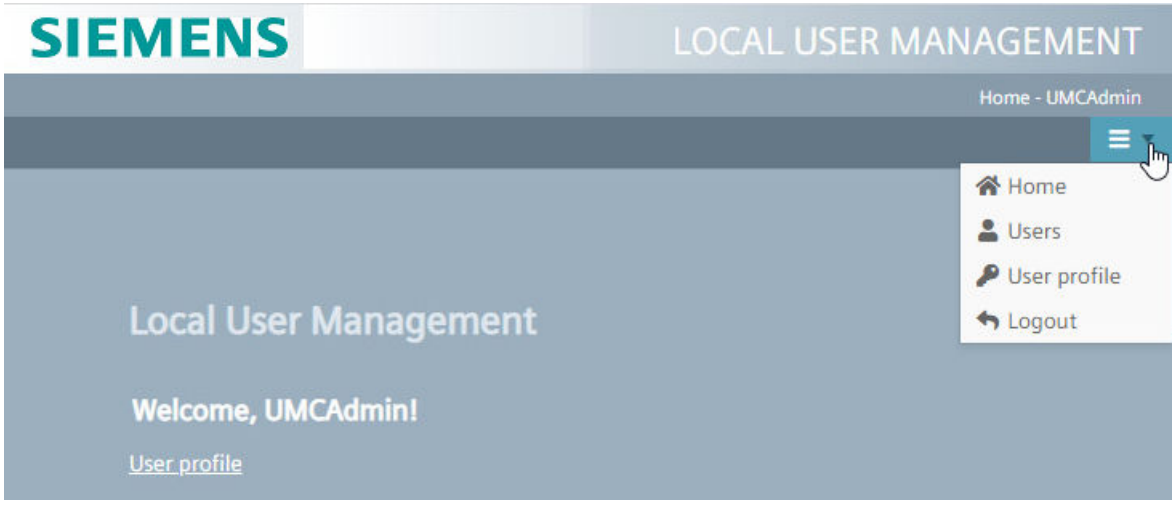
Requirement

- To access the user management, the user must have the "User management" or "Monitor" function right.
- To access the Control Panel, the user must have the "Control Panel access" function right.
- A screen with the "Browser" screen object is displayed in Runtime.

Managing local user in the "Browser" screen object

If you want to manage local users in the "Browser" screen object, follow these steps:

1. Enter the address "https://localhost/umc" in the "Browser" screen object. The "User login" dialog is displayed.
2. Log in to the user management. The home page of the user management opens.



3. Select "Users" in the menu. The user list is displayed.
4. You manage the user information in the user list with the buttons "Add user", "Details", "Edit" and "Delete".

Users without "User management" function right

If you do not have the "User management" function right, you can only select your own "User profile" and change your own password.

13.3.4.4 Opening local user management in the Internet browser

From a PC, you access local user management of a Panel via an Internet browser.

The detailed description of the individual steps in Runtime can be found in the section "Managing local users in Runtime (Page 6930)".

Note

The specific possible operations depend on the function right.

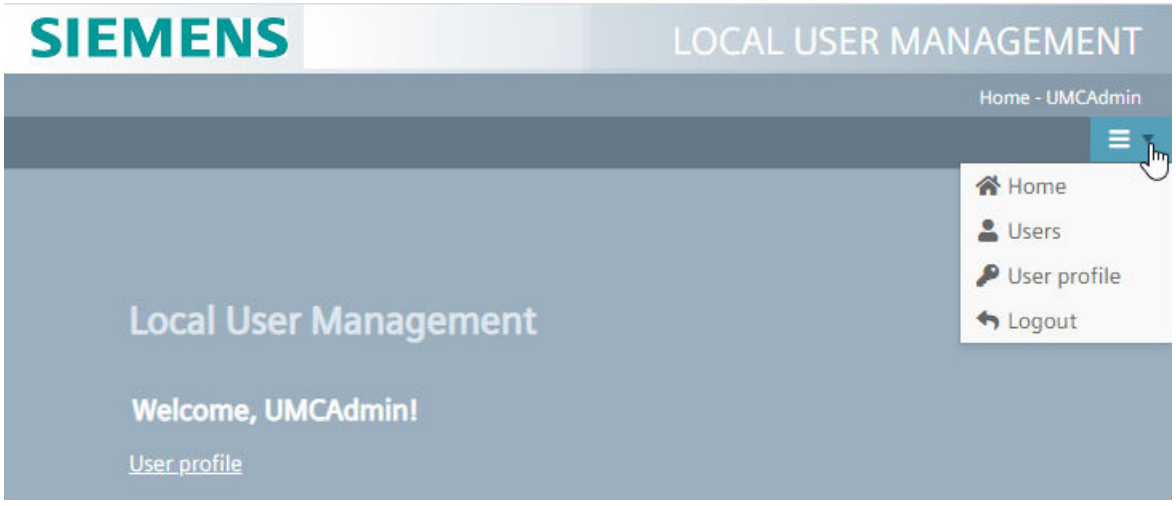
Requirement

- To access the user management, the user must have the "User management" or "Monitor" function right.
- To access the Control Panel, the user must have the "Control Panel access" function right.
- Internet browser is open.

Managing local users in the web browser

If you want to manage local users in the Internet browser, follow these steps:

1. In the address line of the browser, enter the IP address of the Unified Comfort Panel "https://<UCP-IP>/umc". The "User login" dialog is displayed.
2. Log in to the user management. The home page of the user management opens.



3. Select "Users" in the menu. The user list is displayed.
4. You manage the user information in the user list with the buttons "Add user", "Details", "Edit" and "Delete".

Users without "User management" function right

If you do not have the "User management" function right, you can only select your own "User profile" and change your own password.

13.3.4.5 Managing local users in Runtime

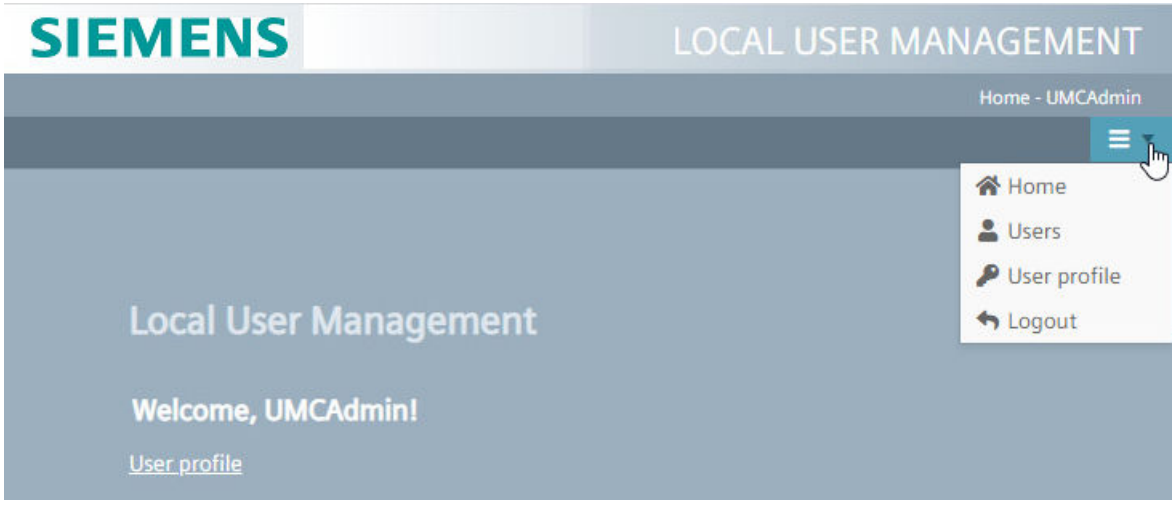
Changing your password

Introduction

You can change your own password in the user management.

Requirement

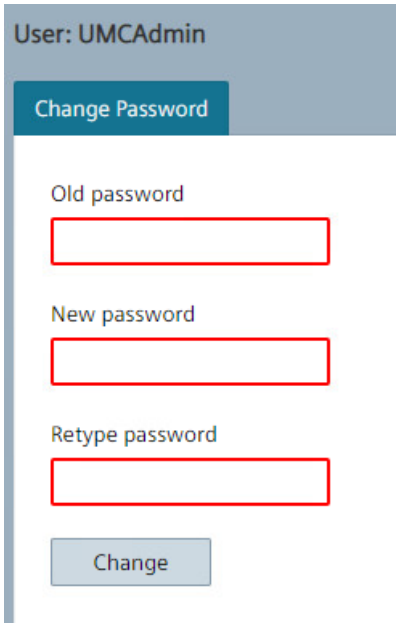
- You have the "User management" function right.
- The home page of the user management is open.



Procedure

To change the password, follow these steps:

1. Select "User profile" directly on the home page or in the menu.
   The "Change Password" dialog is displayed.



2. Change the password and save the change with the "Change" button. The password must meet the password policies.

Changing the password of a different user

Introduction

In the user management you can change the password of a different user. You can also edit the comment.

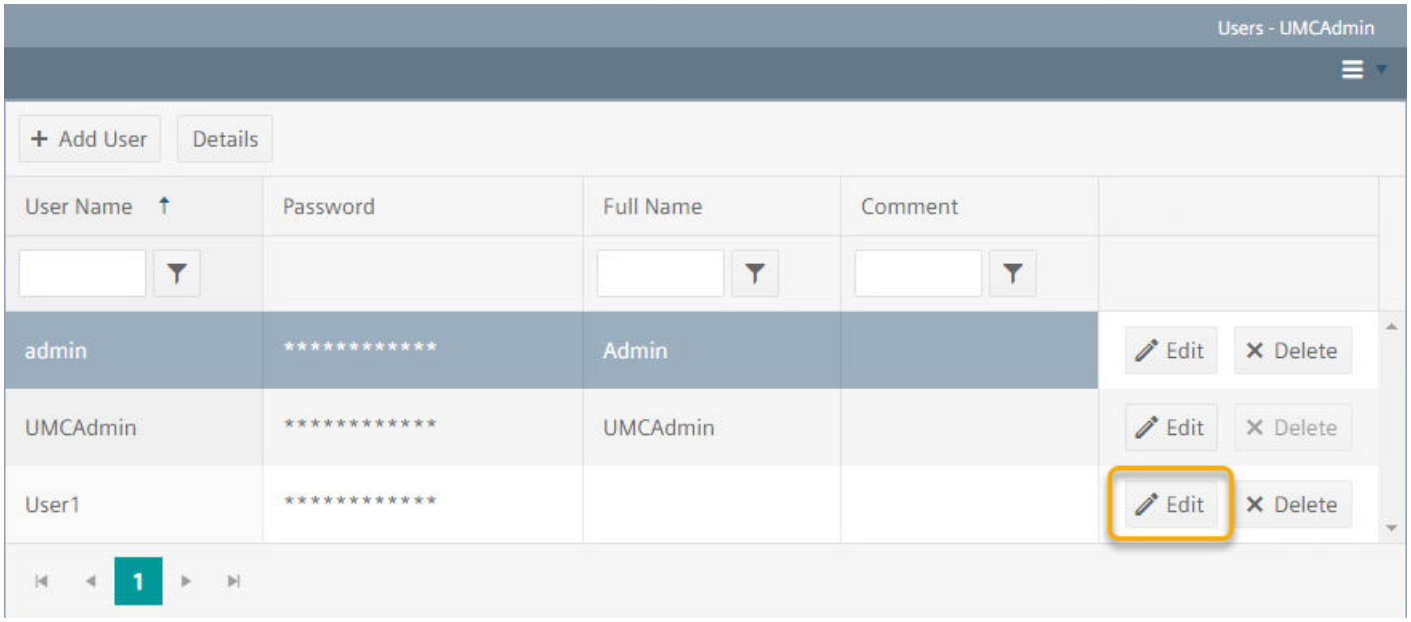
Requirement

- You have the "User management" function right.
- The home page of the user management is open.

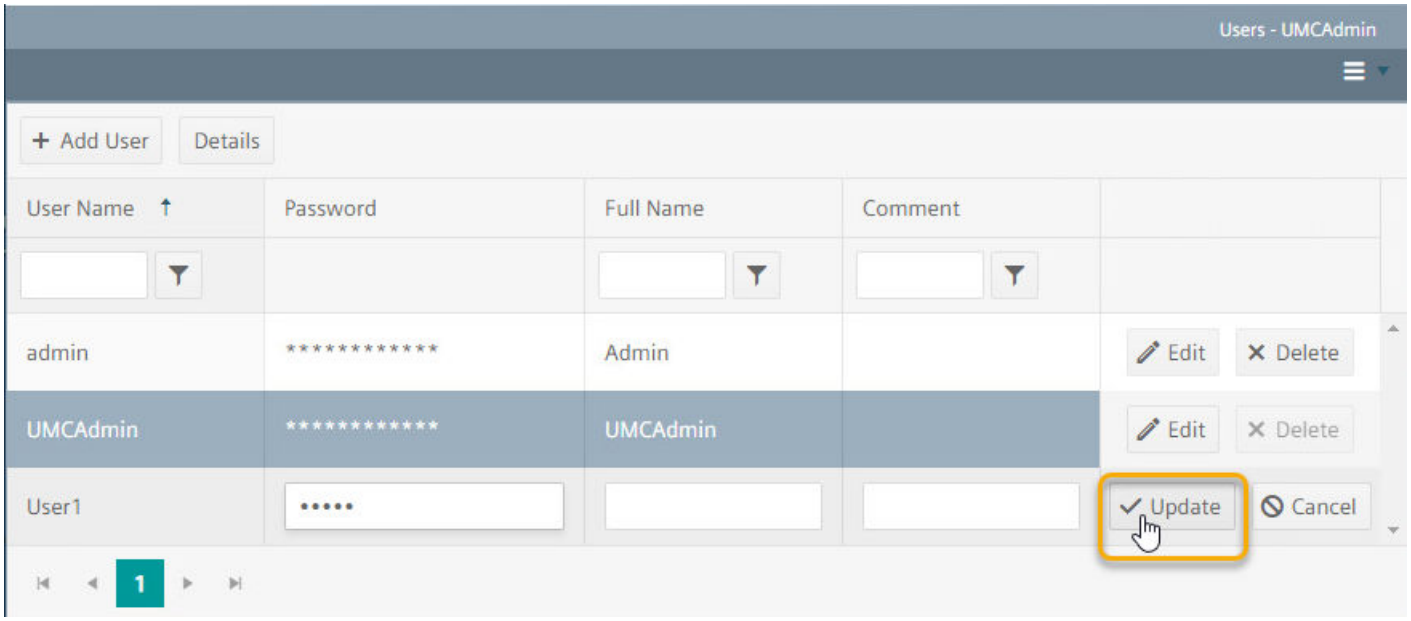
Changing the password and comment

To change the password or comment of a user in the user list, follow these steps:

1. Select "Users" in the menu.
   The user list is displayed.



2. Select a user and click on "Edit" in the respective row.
3. Change the password or the comment and save the change with the "Update" button.



Editing password, status or role

Introduction

In the user list you can edit the password, the status or the role of a user.

Requirement

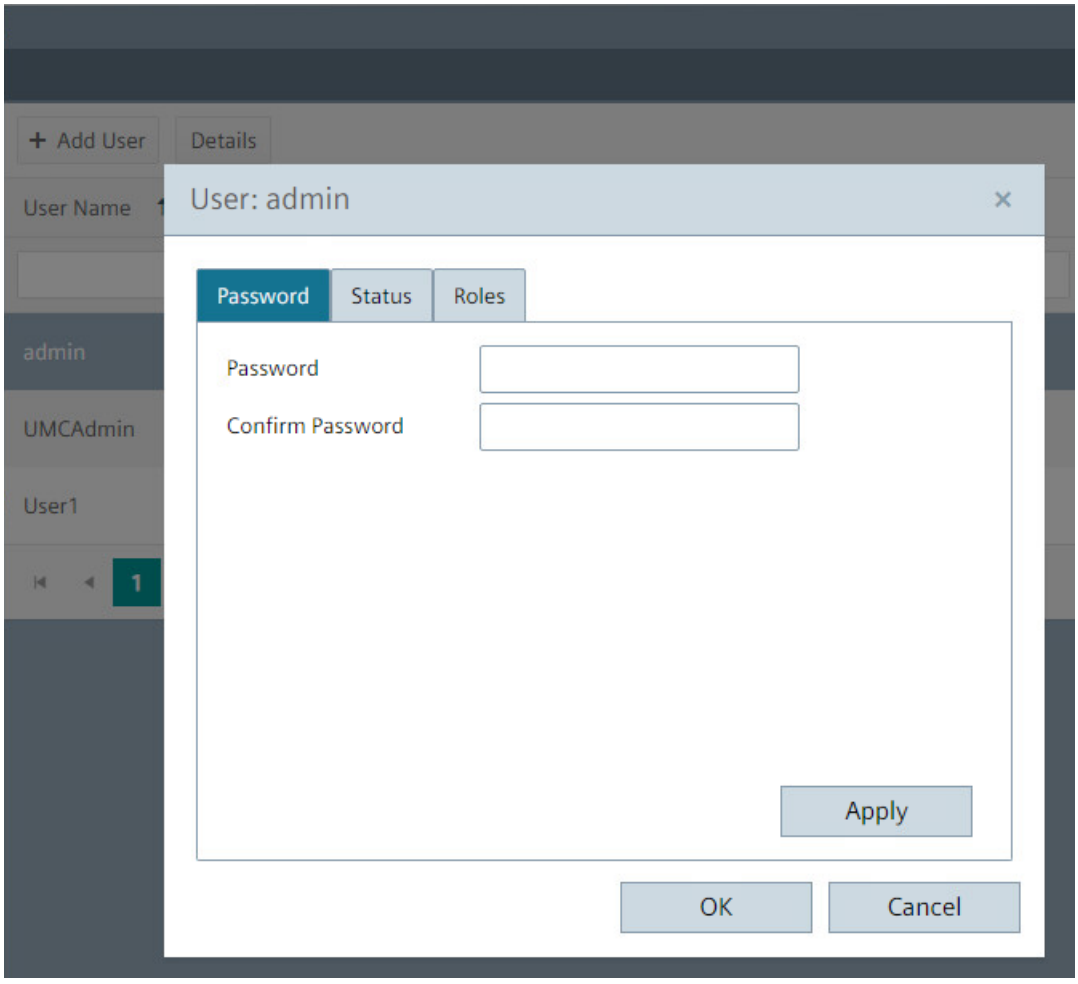
- You have the "User management" function right.
- The home page of the user management is open.

Changing the password

To change the password of a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select a user and click the "Details" button.

3. Enter the new password in the "Password" tab and confirm the password.
4. Confirm your entries with the "Apply" button.
   Save the settings with "OK".

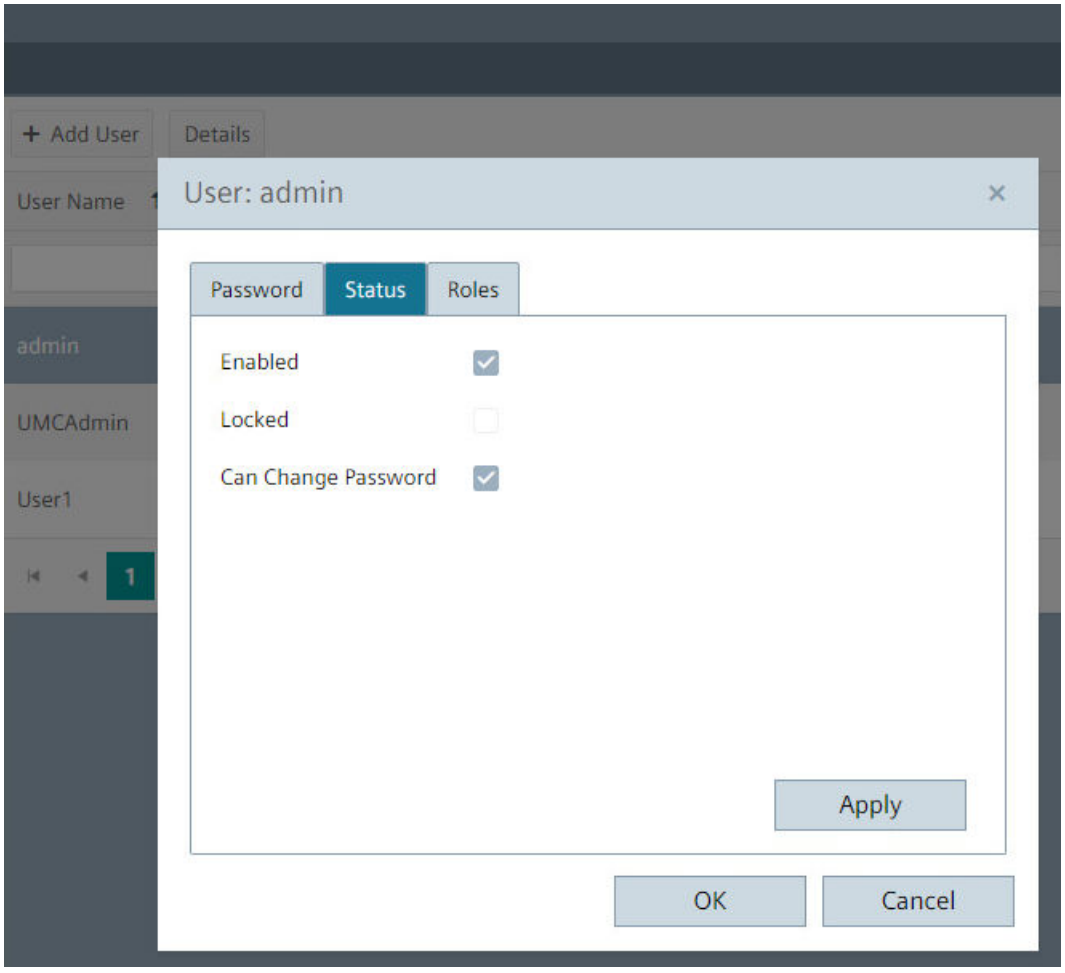


Changing the status

To edit the status of a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select a user and click the "Details" button.

3. In the "Status" tab, you can disable the user or keep this user from changing the password. You cannot change the "Locked" property.
4. Confirm your entries with the "Apply" button.
   Save the settings with "OK".

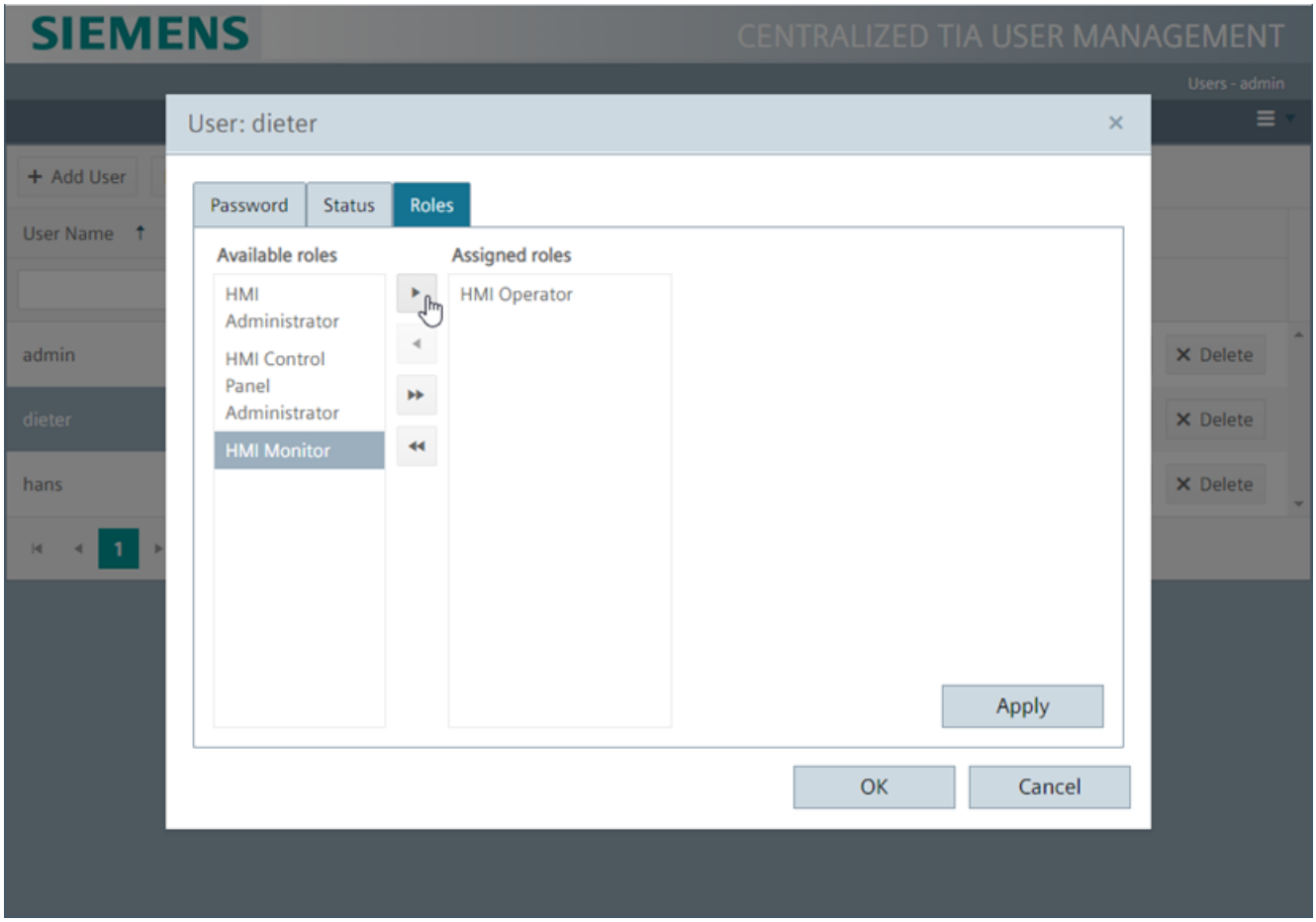


Changing the role

To edit the role of a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select a user and click the "Details" button.
3. In the "Roles" tab, you can change the roles and thus the associated function rights of the user:
   - Select a role from the "Available roles" or "Assigned roles" list.
   - Change the assignment of this role using the buttons between the two lists.
   - Confirm your entries with the "Apply" button.
     Save the settings with "OK".

   The figure below shows you how to assign the "HMI Monitor" role to a user in addition to the "HMI Operator" role.



Note

Note that at least one user in the project has the "HMI Administrator" role and at least one user has access to the Control Panel. If access to the user management or the Control Panel is not possible, a complete download from the TIA Portal is necessary.

Adding users

Introduction

You can add a new user in the user list.

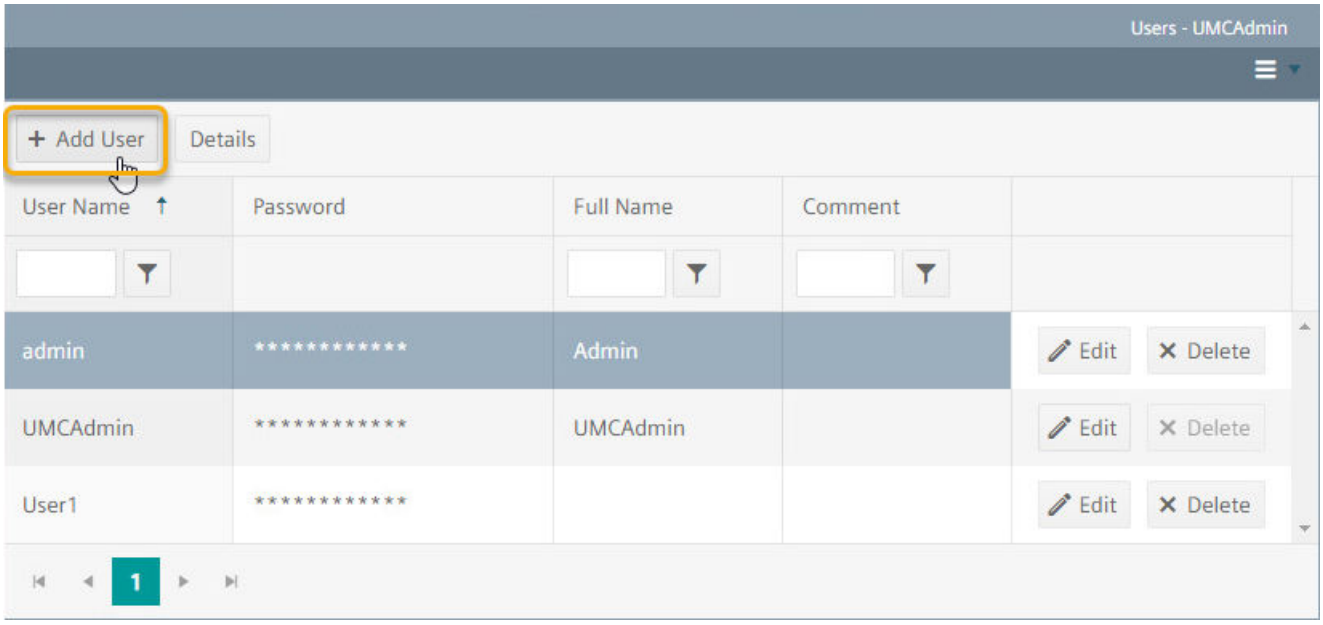
Requirement

- You have the "User management" function right.
- The home page of the user management is open.

Adding a new user

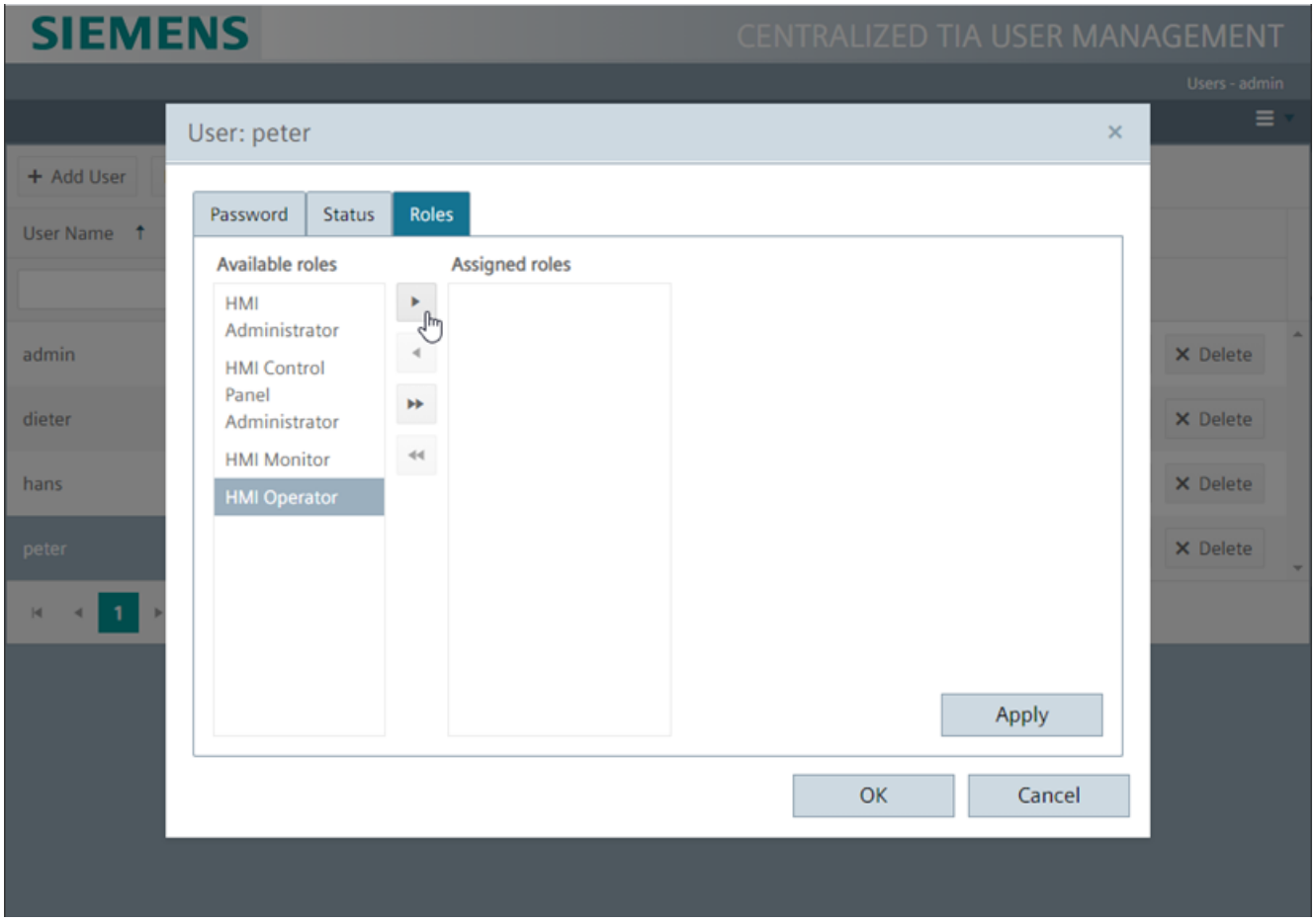
To add a new user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. In the user list, click "Add User".



3. A new row is displayed for the new user in the user list. Enter the information of the new user in the row.

4. Click "Details" in the user list. Assign roles to the new user.



5. Confirm your entries with the "Apply" button.
   Save the settings with "OK".

Deleting users

Introduction

You can delete a user in the user list.

Requirement

- You have the "User management" function right.
- The home page of the user management is open.

Deleting users

To delete a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select the user.
3. Click the "Delete" button in the row. The user is deleted.

Deleting the user from the user list will become effective once the user logs off in Runtime.

13.3.5 Using central user management

13.3.5.1 Using central user management in the Control Panel

In the engineering system you specify whether you want to work with local or central users and user groups from the central user management on a Unified Comfort Panel. By default, the use of the local user management is specified in the engineering system.

Note

You can only switch between local and central user management in the engineering system.

Requirement

- User management has been configured in the engineering system.
- A user with the access right to the Control Panel "Access Control Panel" has been created.
- A Unified Comfort Panel has been started.

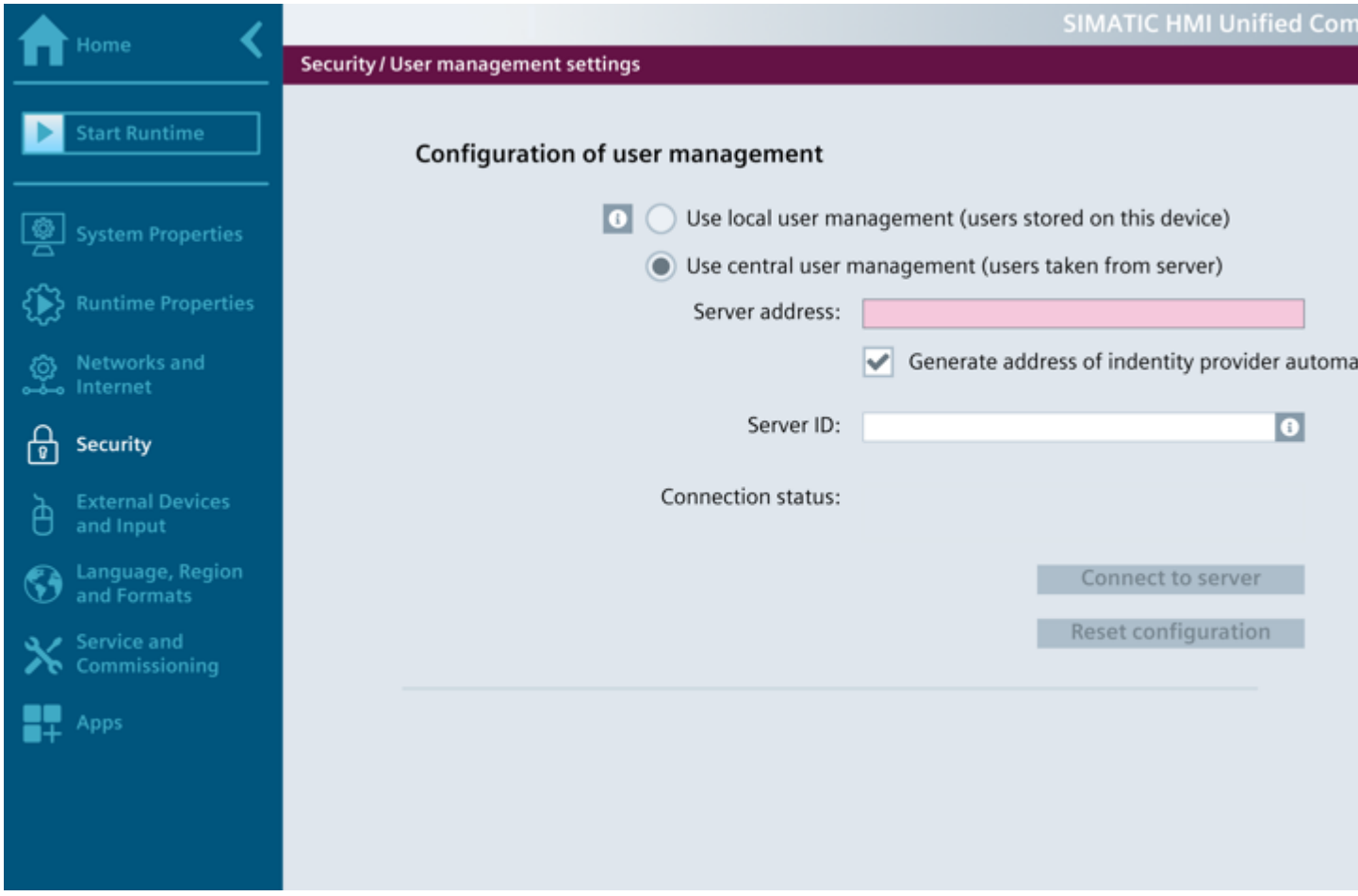
Note

In contrast to the Unified PC Runtime, no login in required. When you open the protected Control Panel, your user rights are checked. If you do not have the necessary rights, a login window is displayed.

Using the user management on the Unified Comfort Panel

If you want to use the user management on the Unified Comfort Panel, follow these steps:

1. Select "Security" in the navigation.
2. Select the "User management settings" tile.
3. In the "Configuration of user management" dialog, you can see whether the local or central user management is enabled. You can only switch between local and central user management in the engineering system.



Establishing a connection to the central user management

If you have not entered any information in the engineering system, follow these steps:

1. Enter the server address.
2. The "Connect to server" button is activated.
3. Enter the server ID manually or click the "Connect to server" button. The server ID is read in from the central user management.
4. In the "Verify user management server" dialog, confirm the displayed server.
   All input fields are filled in and write-protected. Status of the connection is "Connected".
   - The "Connect to server" button is changed to "Check connection".
   - You can check or reset the connection.

Adapting the connection to the central user management

If you have only entered some information in the engineering system, the input field for the missing information is marked in red. Proceed as follows in this case:

1. Enter the missing information.
2. The "Connect to server" button is activated.
3. Click "Connect to server". The server ID is read in from the central user management.
4. In the "Verify user management server" dialog, confirm the displayed server.
   All input fields are filled in and write-protected. Status of the connection is "Connected".
   - The "Connect to server" button is changed to "Check connection".
   - You can check or reset the connection.

If the entered server ID is not the same as the online server ID, you can apply the online server ID.

1. Confirm this in the "Verify user management server" dialog.
   The online server ID is read in. All input fields are filled in and write-protected. Status of the connection is "Connected".
   - The "Connect to server" button is changed to "Check connection".
   - You can check or reset the connection.

Verifying the connection to the central user management

You have entered all information in the engineering system. When the server ID in the engineering system and on the Panel match, the Panel is automatically connected to the central user management.

Status of the connection is not displayed. If you want to verify the central user management on the Unified Comfort Panel, follow these steps:

1. Click "Check connection".
2. The connection to the central user management is established.

When the Panel is not connected to the central user management and no status is displayed, change the server ID.

1. The "Check connection" button is changed to "Connect to server".
2. You can make or reset the connection.

Resetting the connection to the central user management

If you have entered all information in the engineering system, the Panel is connected to the central UMC server. If you want to reset the central user management, follow these steps:

1. Click "Reset configuration".
2. In the "Reset central user management" dialog, confirm the reset.
3. The default is set:
   - The input field for the server address is empty and marked in red.
   - The input field for the server ID is empty.
   - The "Connect to server" button is hidden.
   - The "Reset configuration" button is hidden.

13.3.5.2 Simulating a central user management

You want to simulate a project in which a central user management is configured for a customer. You have two options if you do not have access to the central user management of the customer:

- Configure your own central user management.
- Configure a local user management.

Requirement

- You know which groups and their function rights are contained in the central user management of the customer.

Configuring a central user management

Configure a central user management for the simulation project.

1. Create the users.
2. Create the user groups according to the customer project.
3. Assign the users to the groups.
4. Establish the connection to the central user management.
5. Start the simulation.
6. Log on in runtime.

Changes can be downloaded.

Configuring a local user management

Configure a local user management.

1. Create one or more users.
2. Assign the roles to the users.
3. Start the simulation.

Changes cannot be downloaded.

See also

Simulate Unified Comfort Panel (Page 7205)

13.4 Using user management on the WinCC Unified PC

13.4.1 Notes on commissioning

Commissioning with central user management

To establish the connection to the central user management, configure the connection to the central user management on the PC with SIMATIC Runtime Manager.

Note

Failure of the network connection

If the network connection to the central user management fails, all users who have already logged on to the project once before can still log on for some time. When the connection to the central user management has been restored, other users can log on once again as well.

See also

SIMATIC Runtime Manager users (Page 6947)

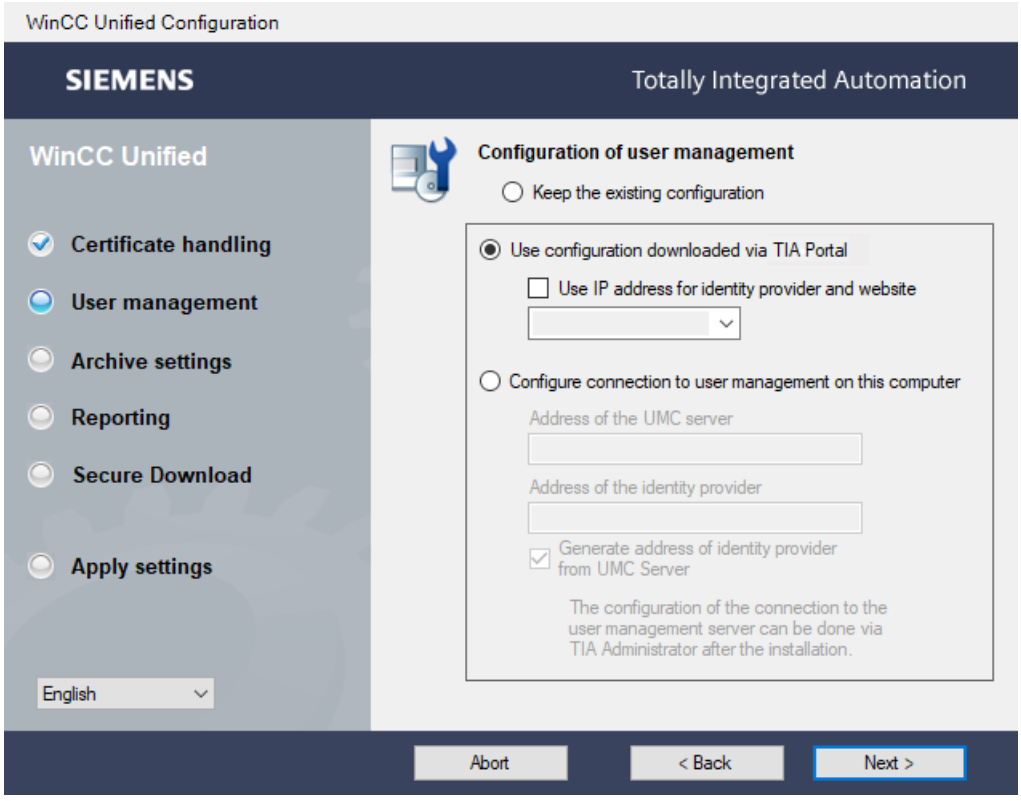
13.4.2 Setting the user management with WinCC Unified Configuration

The "WinCC Unified Configuration" application is installed with the installation of WinCC Unified. A link is automatically created on your desktop for "WinCC Unified Configuration".

You can change the setting made during setup at any time.

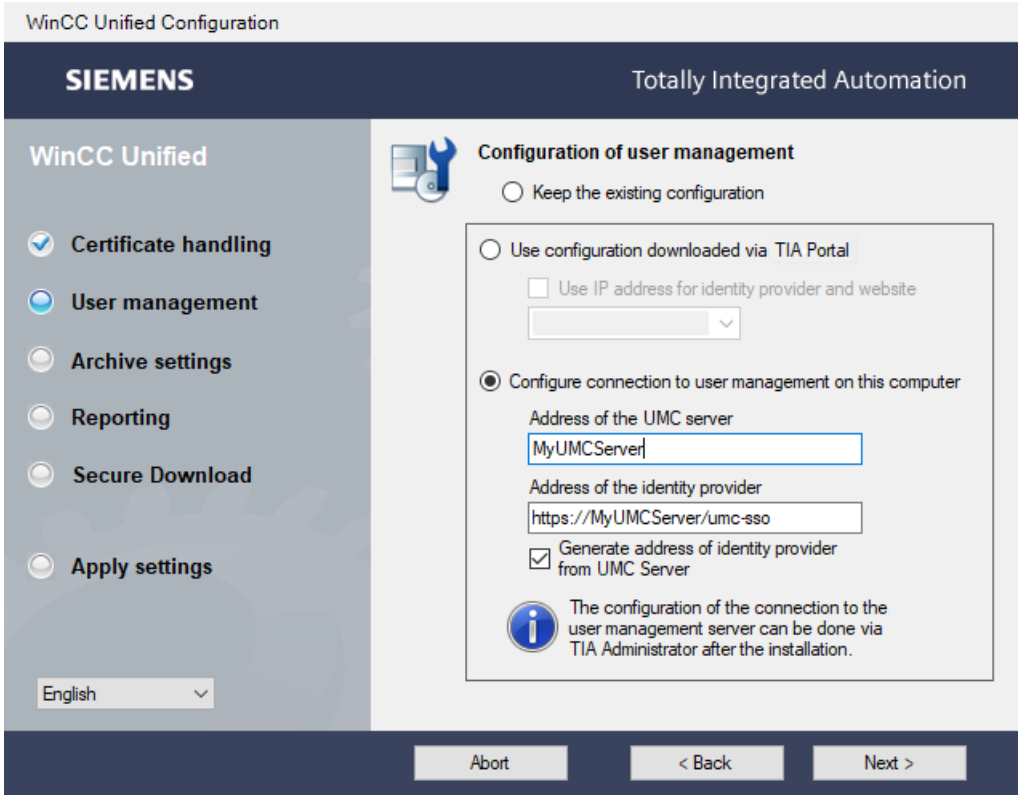
Local user management

If you only edit projects with local user management in the TIA Portal, use the setting "Use configuration downloaded via TIA Portal".



Central user management

If you only edit projects with central user management in the TIA Portal, use the setting "Configure connection to user management on the computer".



Using local and central user management

When you edit projects with local as well as central user management, note the following special features:

Setting "Use configuration downloaded via TIA Portal":

- You cannot simulate projects with central user management.
- Groups and users of the central server cannot be imported into projects with central user management.

Setting "Configure connection to user management on the computer":

- You cannot simulate projects with local user management.

13.4.3 Managing multiple projects in the SIMATIC Runtime Manager

The SIMATIC Runtime Manager is installed with the installation of WinCC Unified Runtime. A link for the SIMATIC Runtime Manager is automatically created on your desktop.

You manage the Runtime projects in the SIMATIC Runtime Manager. Multiple projects can be available on one PC. In SIMATIC Runtime Manager, you enable the matching configuration of the user data for a selected project or you adapt the configuration.

You cannot switch between local and central user management in the Runtime Manager.

See also

Checking local user management in the SIMATIC Runtime Manager (Page 6947)

Setting central user management in the SIMATIC Runtime Manager (Page 6966)

13.4.4 SIMATIC Runtime Manager users

Groups for the user management

Groups are created during the installation of WinCC Unified that enable a user to access specific WinCC Unified functions:

- **umcd_domain_manager**
- **umcd_um**
- **umcd_dsso**

To log in with single sign-on (SSO), the user must be added to the following group:

- **umcd_dsso**

Groups for the Runtime Manager

For a Windows user to be able to change user data in the Runtime Manager, add the user to the following groups:

- **umcd_domain_manager**
- **umcd_um**
- "SIMATIC HMI"

See also

Notes on commissioning (Page 6944)

13.4.5 Managing local users

13.4.5.1 Checking local user management in the SIMATIC Runtime Manager

Multiple projects can be available on one PC. In SIMATIC Runtime Manager, you can activate the matching configuration of the user data for a selected project.

You cannot switch between local and central user management.

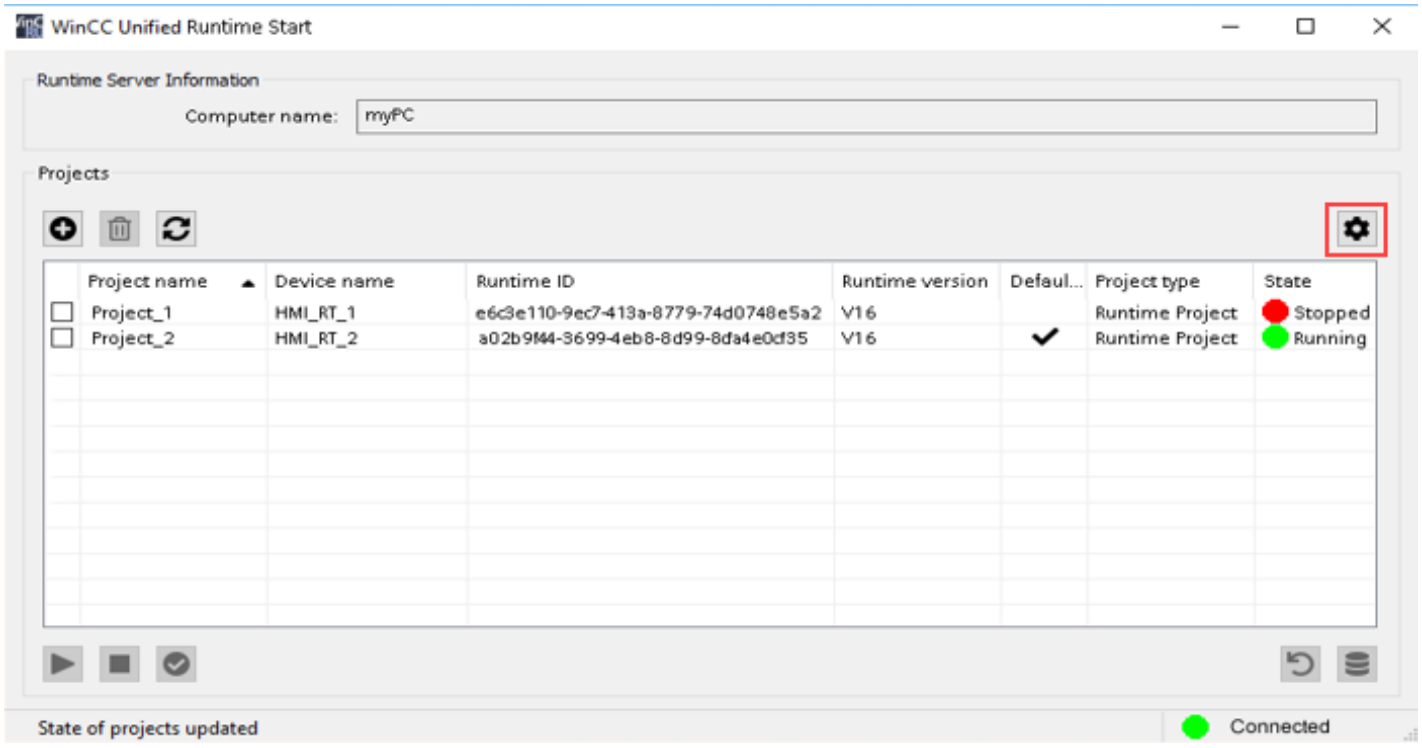
Requirement

- At least one project with user management is loaded on the Runtime PC.
- WinCC Unified Runtime is installed.
- SIMATIC Runtime Manager has been started.

Checking local user management in the SIMATIC Runtime Manager

You cannot configure the local user management in the SIMATIC Runtime Manager.

1. In the SIMATIC Runtime Manager, click "Settings" on the home page.



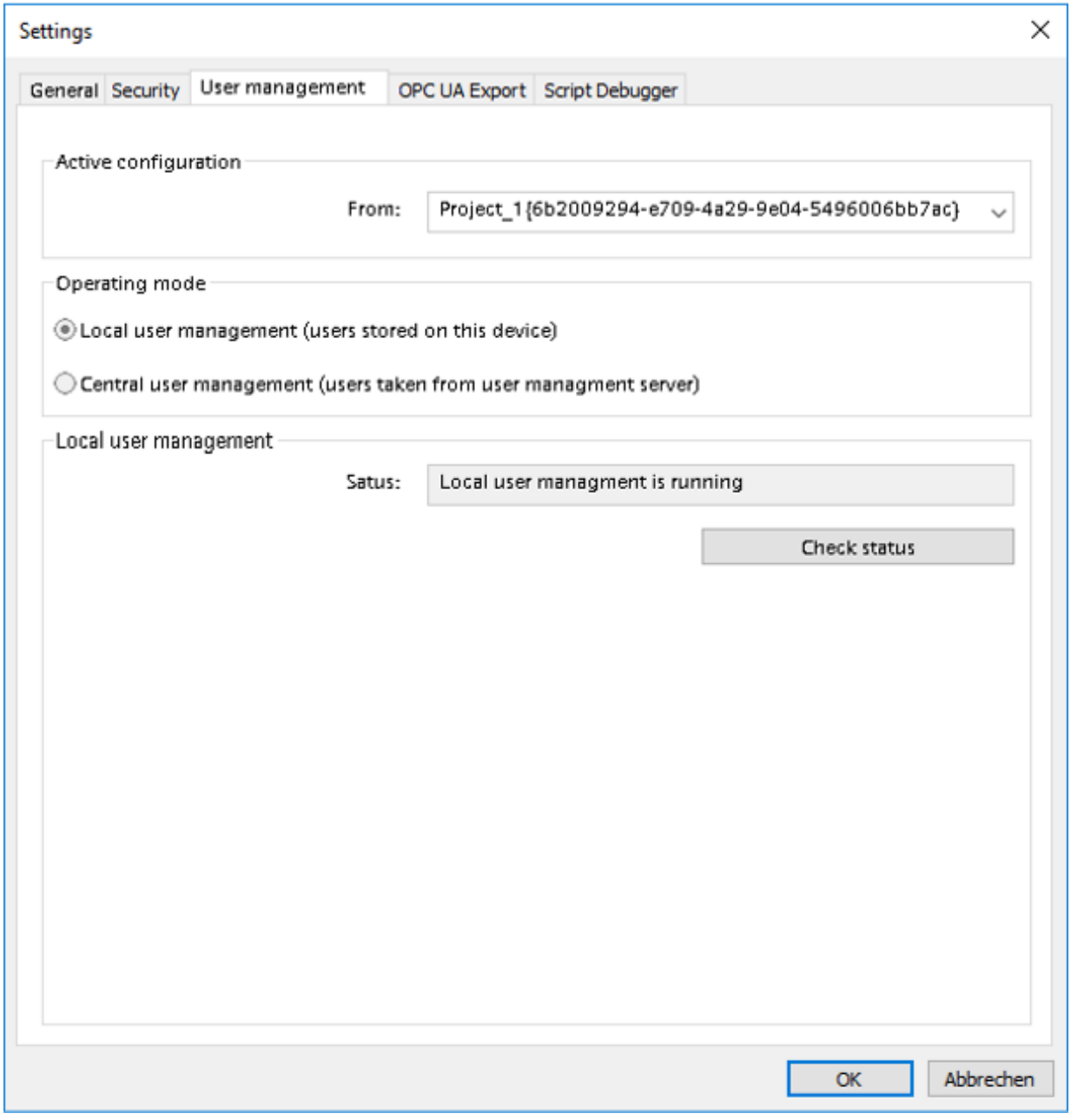
2. Select the "User management" tab.
3. When multiple projects are available on the PC, select the configuration of the user management of a project under "Select configuration".

4. To activate the local user data of the selected project, click "Load user management".

   Note

   When you click "Load user management", the changes of the user management that you have made in runtime are overwritten by the configuration of the last download.

5. The status of the local user management is displayed. You check the status of the local user management with the "Check status" button.



See also

Managing multiple projects in the SIMATIC Runtime Manager (Page 6946)

13.4.5.2 Managing local users in Runtime

User logon

Introduction

From a PC, you access local user management via an Internet browser.

To manage the local users on a Unified PC, you require the "User management" function rights. Configure a user with the required rights in the engineering system and load the user into Runtime.

Note

The specific possible operations depend on the function right.

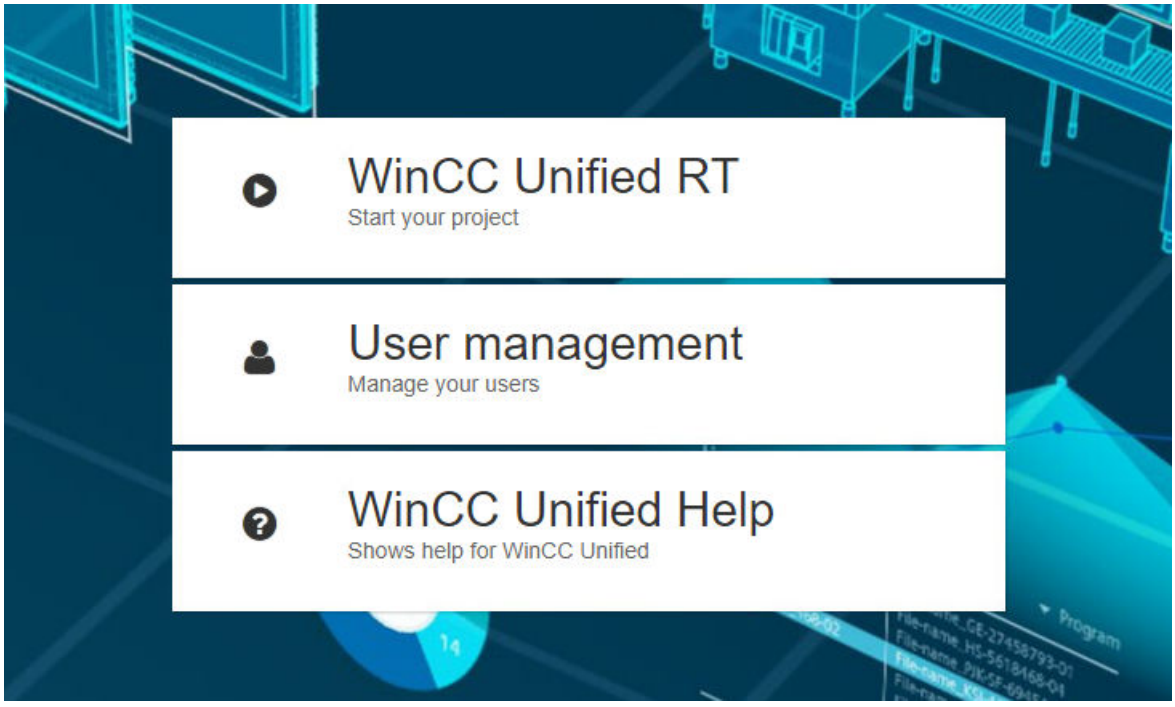
Requirement

- The user has the "User management" function rights.
- Internet browser is open.

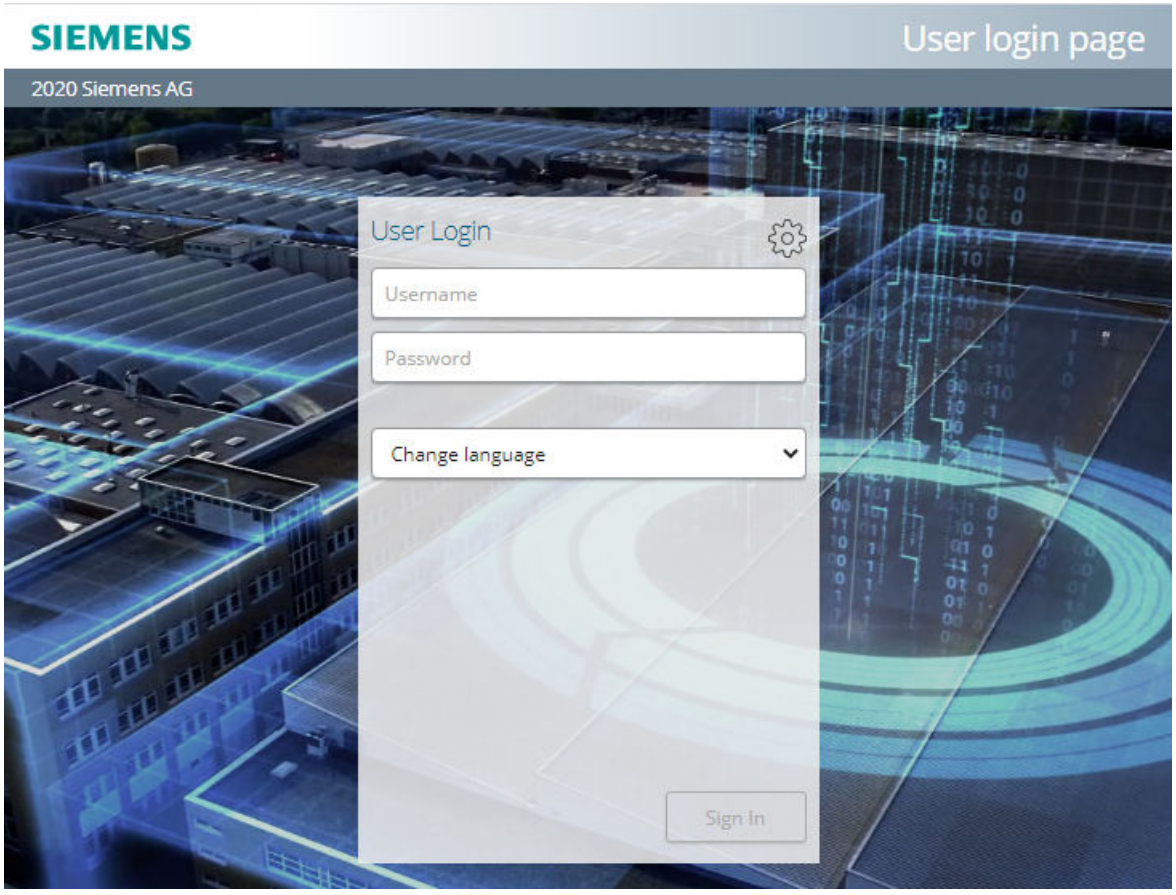
Logging on to the user management

To log on to the user management in Runtime, follow these steps:

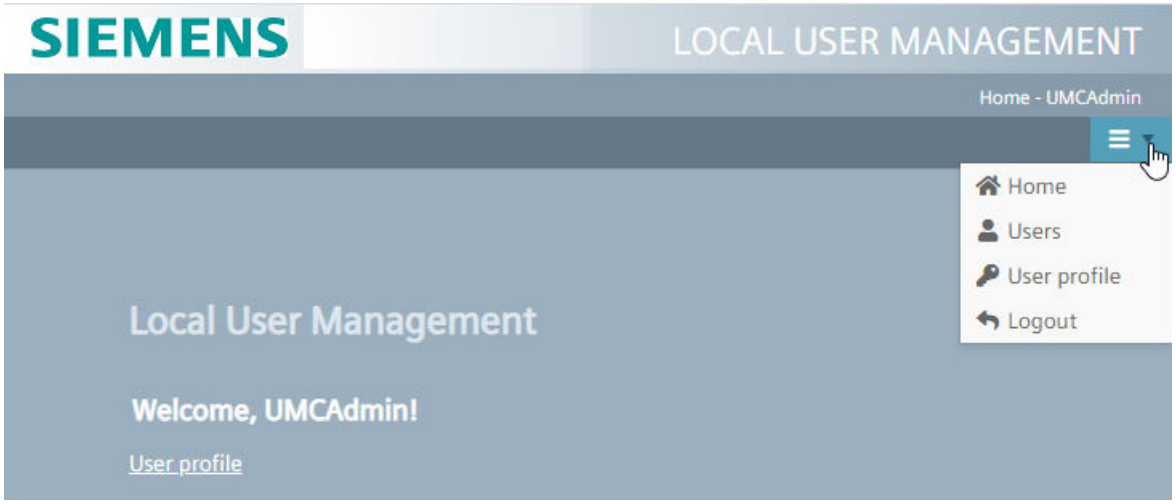
1. In the browser, enter the IP address of the Runtime PC "https://<PC-IP>/umc". If runtime is installed on the same PC as the browser, enter the address "https://localhost/umc". The start page of Runtime is displayed.



2. Click the "User management" button. The "User login" dialog is displayed.



3. Type in the user name and password.
4. If required, use the selection list to change the displayed language.
5. Click "Login".
   The user management start page opens in Runtime.



See also

Managing local users in Runtime (Page 6950)

Structure of the start page

Introduction

In menu on the start page, select whether you want to manage the users, change the password or language, or log out. You can find the menu via the drop-down list in the upper right corner.

Note

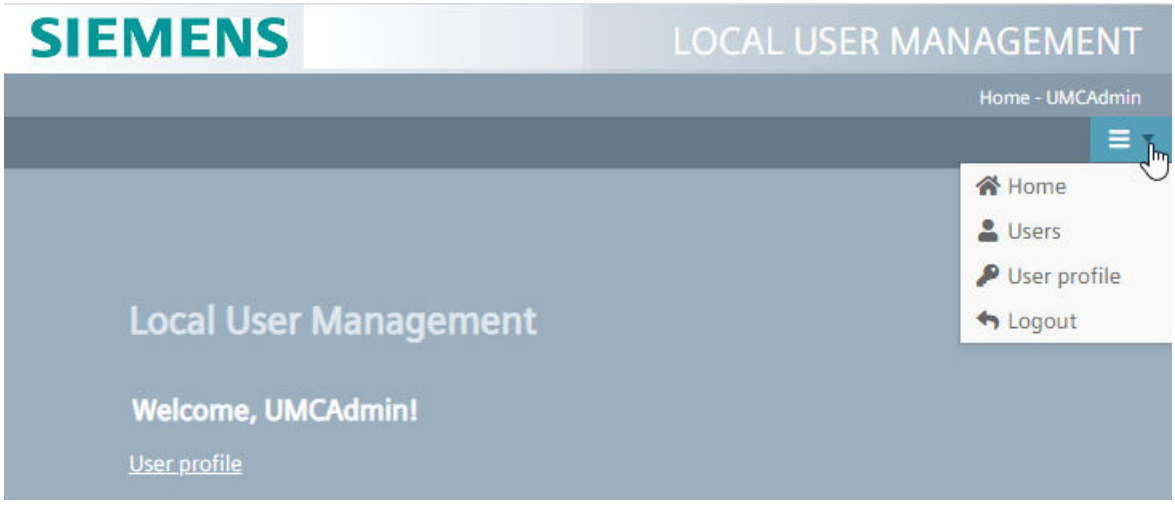
Users with the "User management" function right have access to all functions.

Users without the "User management" function right can change their password under "User profile".

Menu

The following options are available to you under the symbols in the menu:

- "Home"
  This takes you to the start page of the user management.
- "Users"
  You can create new users or manage the existing users.
- "User profile"
  You can change your password and switch the language.
- "Logoff"
  You will be logged out directly and can log in again.



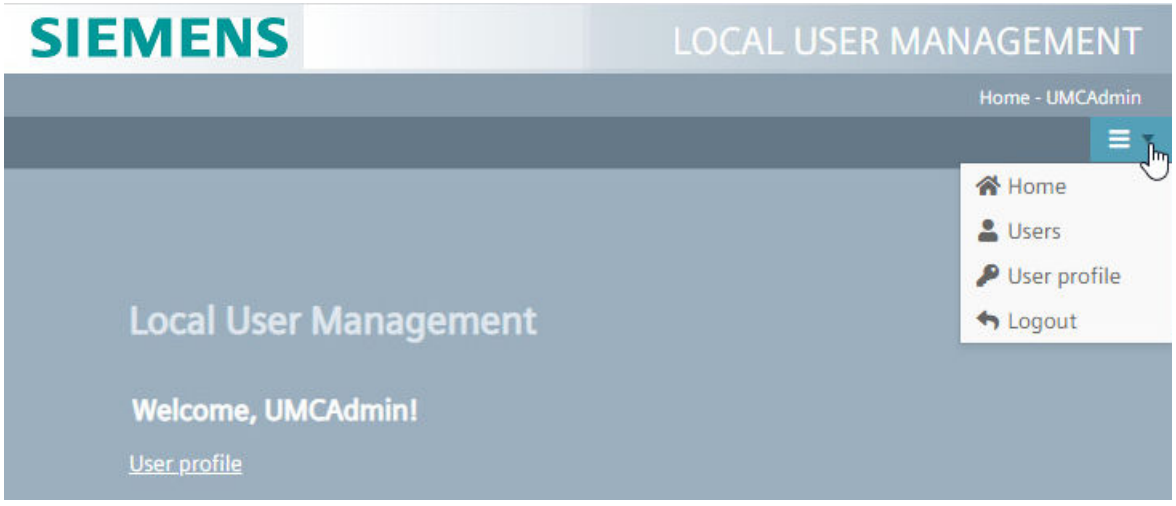
Changing your password

Introduction

You can change your own password in the user management.

Requirement

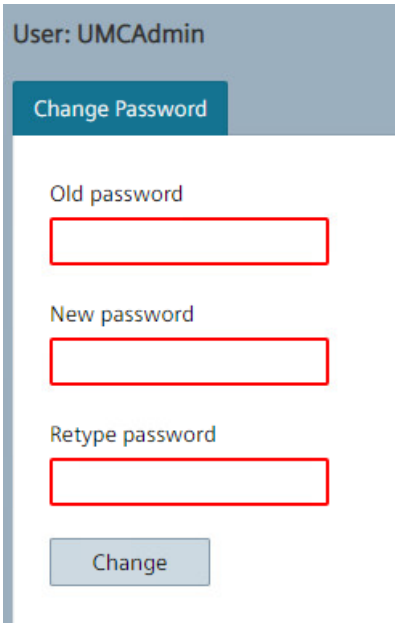
- You have the "User management" function right.
- The home page of the user management is open.



Procedure

To change the password, follow these steps:

1. Select "User profile" directly on the home page or in the menu.
   The "Change Password" dialog is displayed.



2. Change the password and save the change with the "Change" button. The password must meet the password policies.

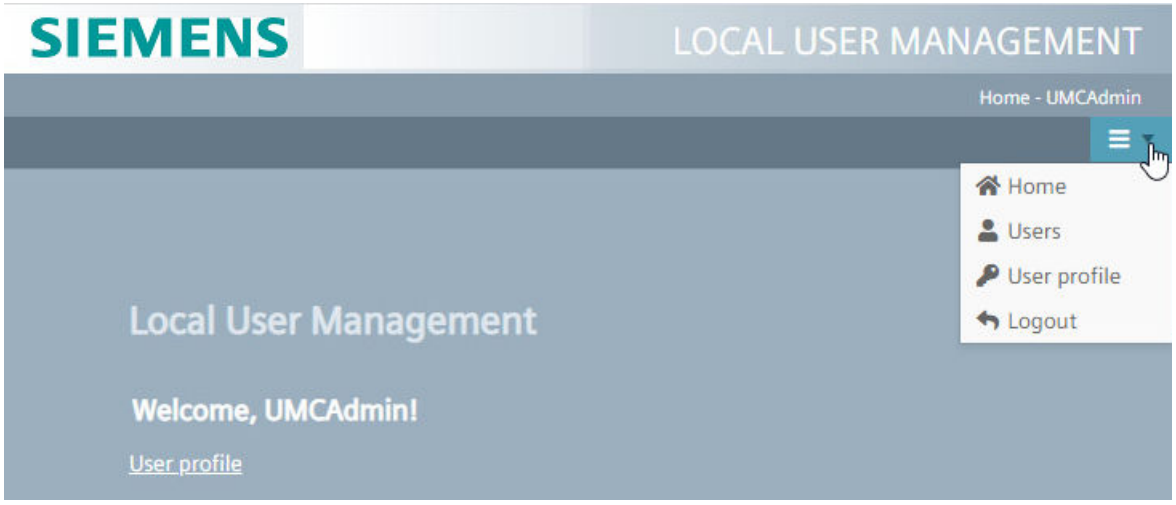
Managing the user list

Introduction

In the user list you can manage the data of the other users.

Requirement

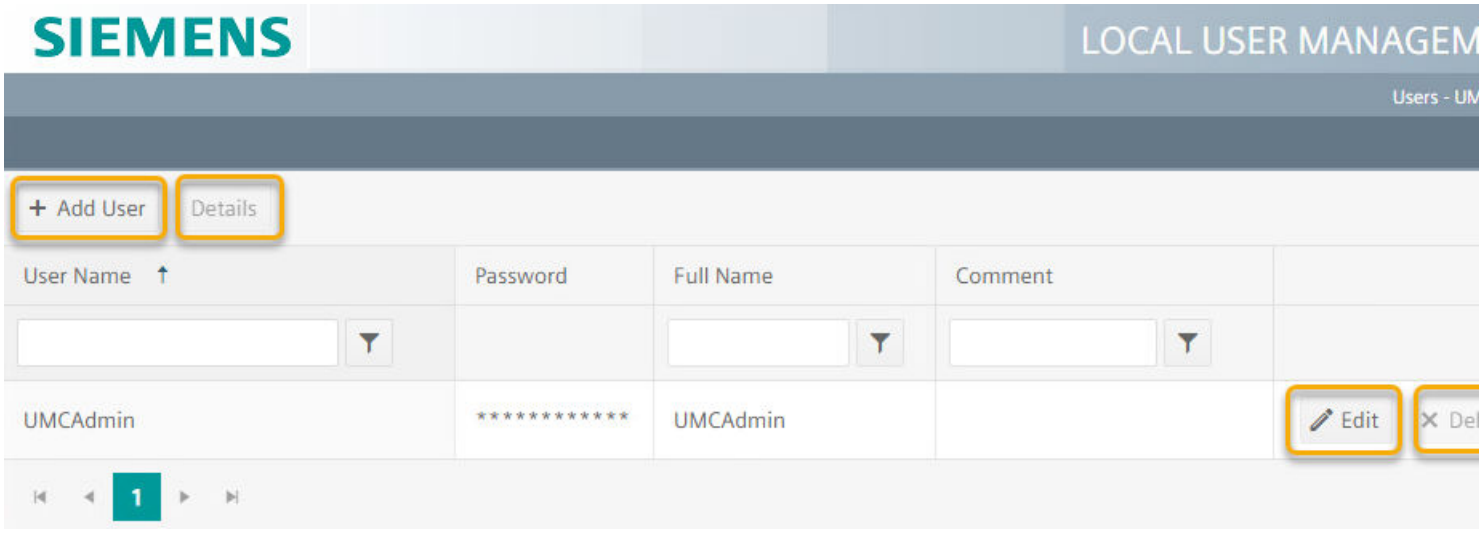
- You have the "User management" function right.
- The home page of the user management is open.



Opening a user list

To display the user list, click "Users" in the menu on the homepage.

The user list is displayed.



Options in the user list

In the user list you can manage the data of the user via the following buttons:

- "Add users"
- "Details"

- "Edit"
- "Clear"

In the user list, you can:

- Sort users by user name or comment.
- Filter users by user name or comment.
- Display 20 users on one page. Additional users are displayed on a new page. You can switch between the pages.

Changing the password of a different user

Introduction

In the user management you can change the password of a different user. You can also edit the comment.

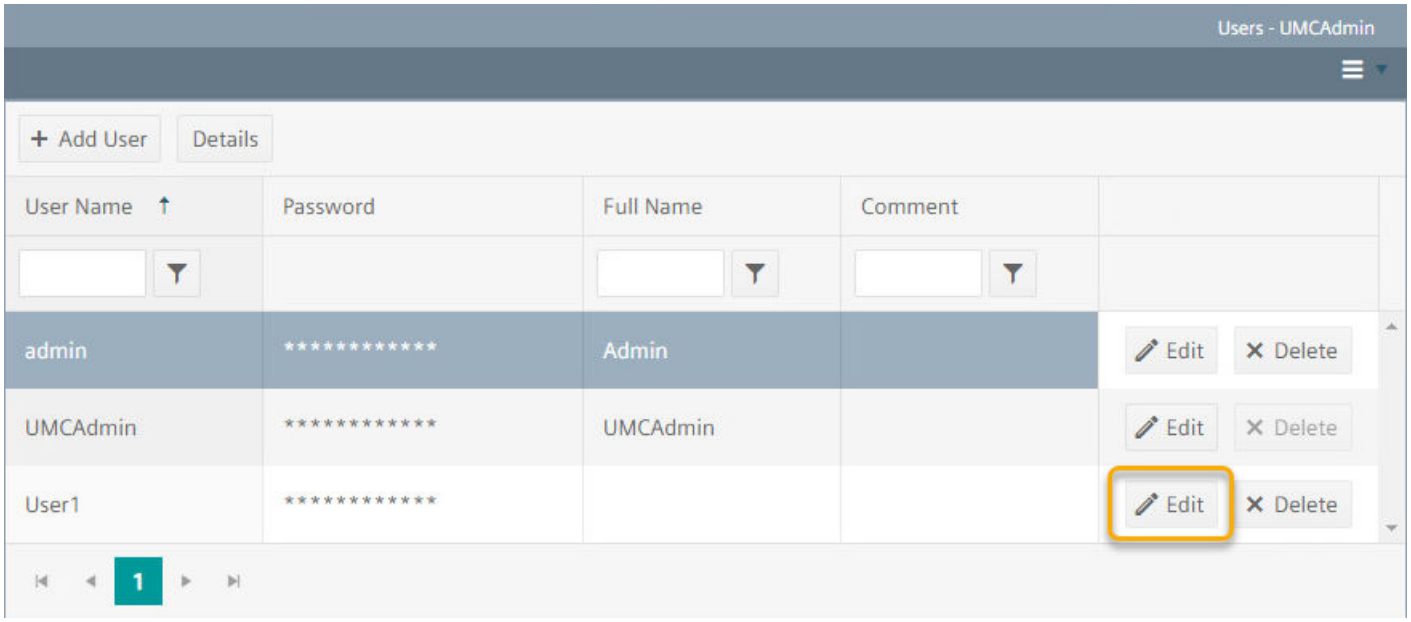
Requirement

- You have the "User management" function right.
- The home page of the user management is open.

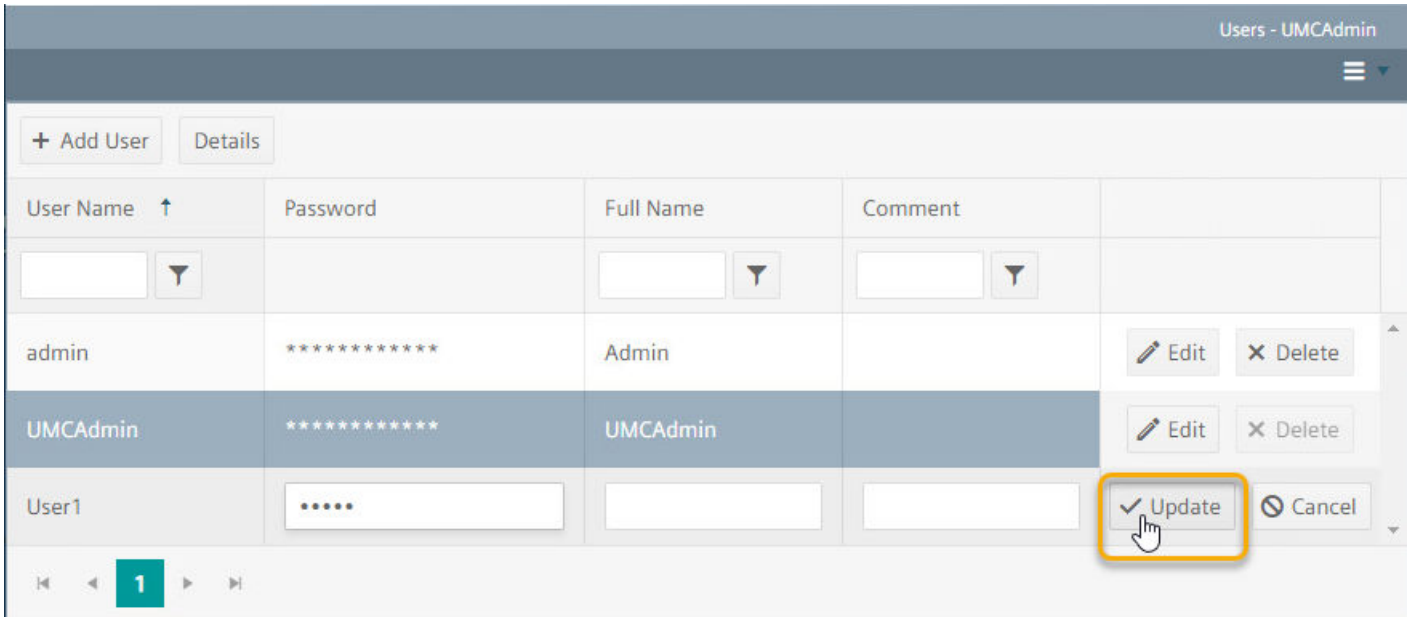
Changing the password and comment

To change the password or comment of a user in the user list, follow these steps:

1. Select "Users" in the menu.
   The user list is displayed.



2. Select a user and click on "Edit" in the respective row.
3. Change the password or the comment and save the change with the "Update" button.



Editing password, status or role

Introduction

In the user list you can edit the password, the status or the role of a user.

Requirement

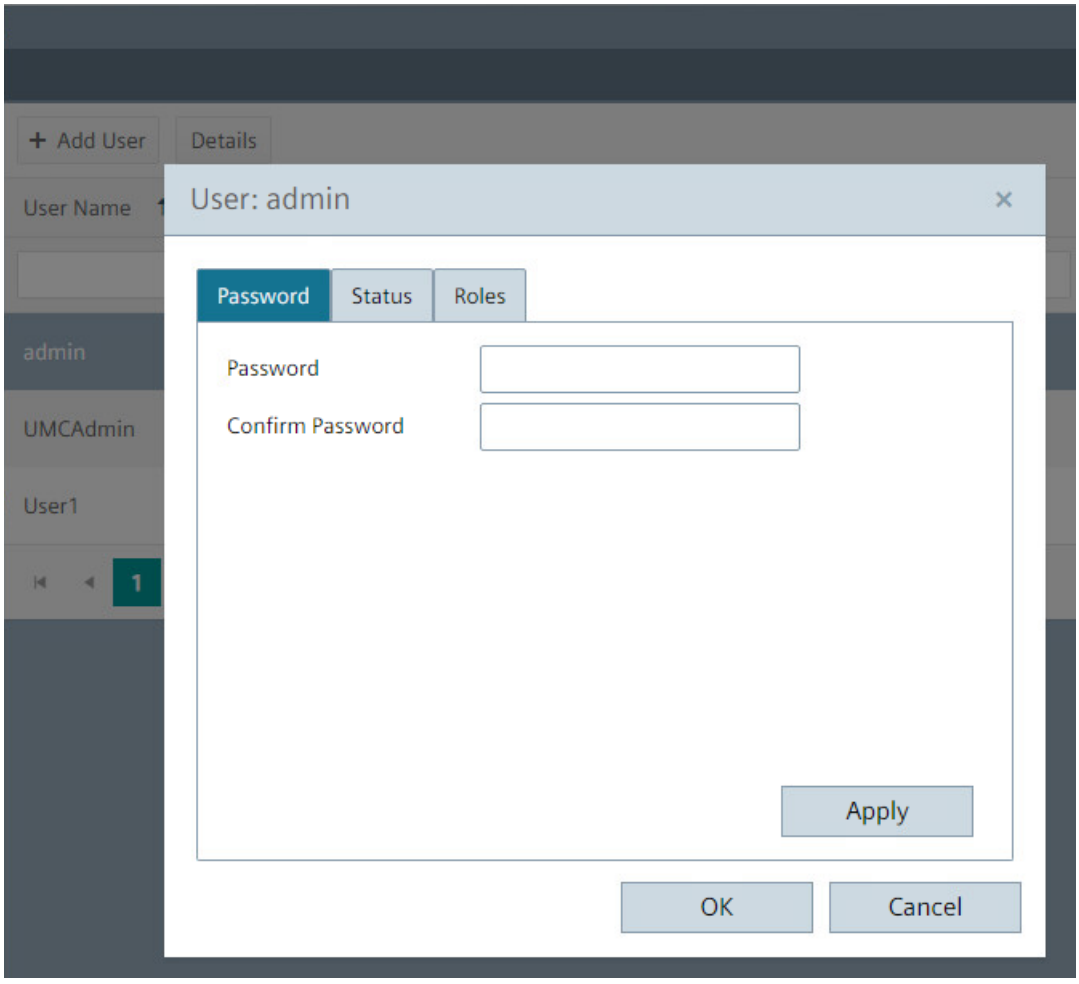
- You have the "User management" function right.
- The home page of the user management is open.

Changing the password

To change the password of a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select a user and click the "Details" button.

3. Enter the new password in the "Password" tab and confirm the password.
4. Confirm your entries with the "Apply" button.
   Save the settings with "OK".

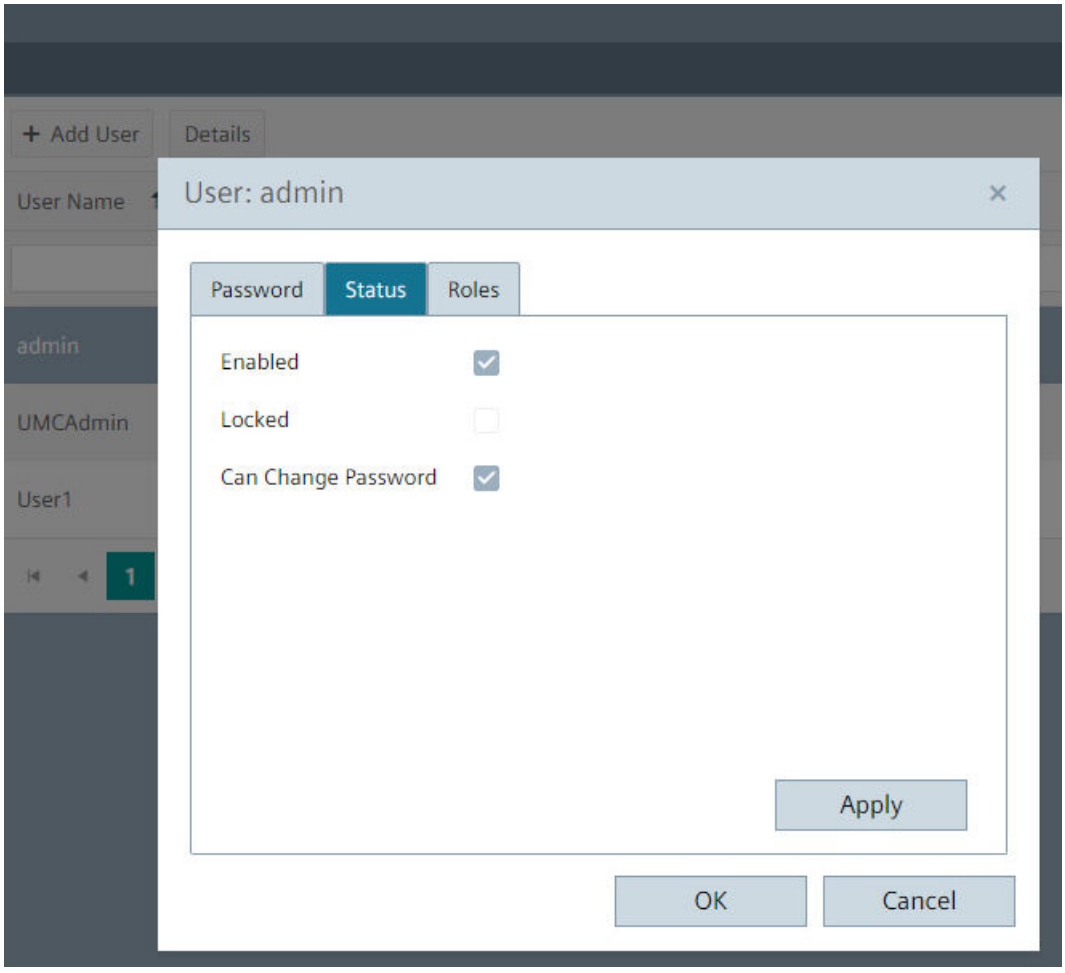


Changing the status

To edit the status of a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select a user and click the "Details" button.

3. In the "Status" tab, you can disable the user or keep this user from changing the password. You cannot change the "Locked" property.
4. Confirm your entries with the "Apply" button.
   Save the settings with "OK".

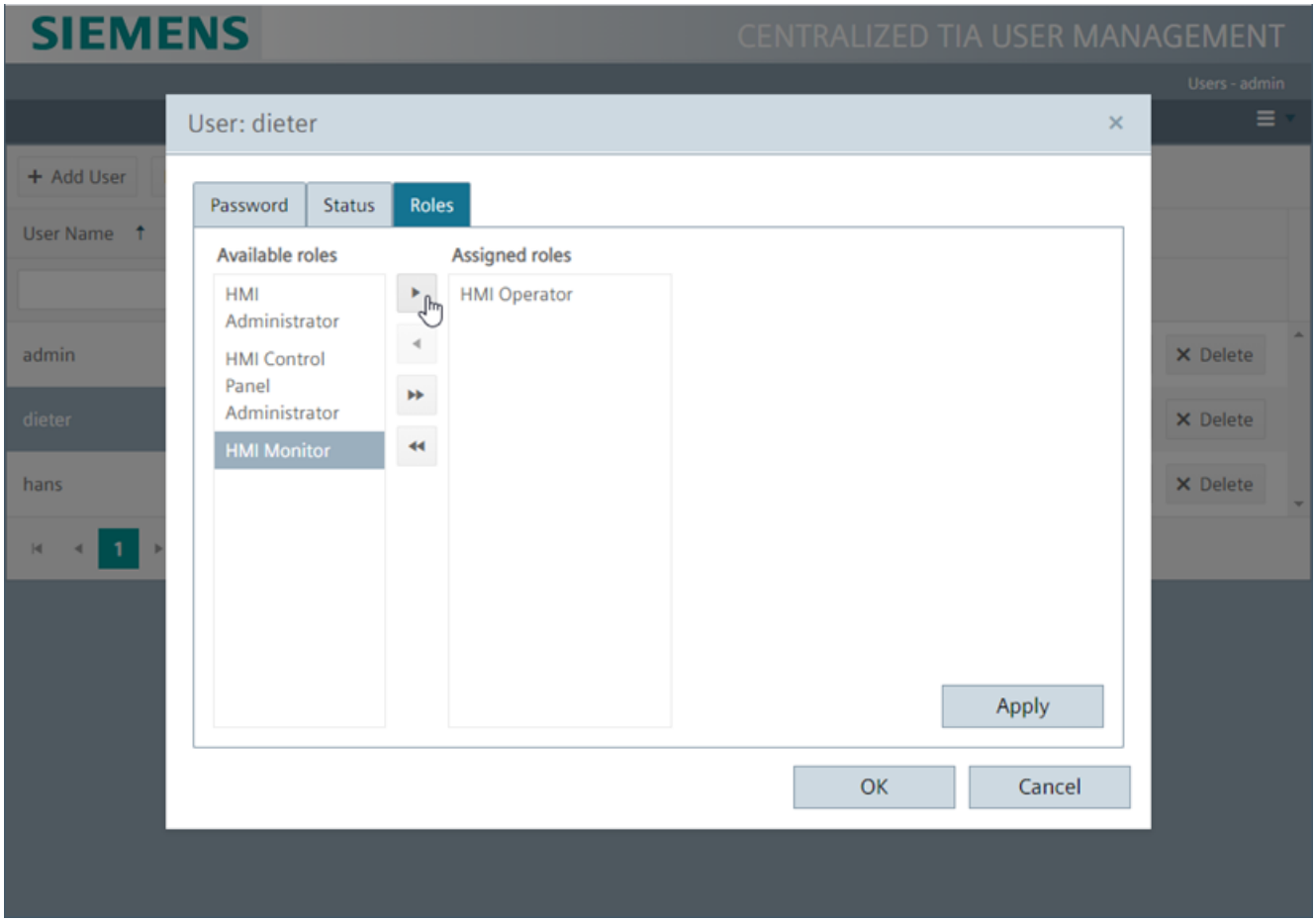


Changing the role

To edit the role of a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select a user and click the "Details" button.
3. In the "Roles" tab, you can change the roles and thus the associated function rights of the user:
   - Select a role from the "Available roles" or "Assigned roles" list.
   - Change the assignment of this role using the buttons between the two lists.
   - Confirm your entries with the "Apply" button.
     Save the settings with "OK".

   The figure below shows you how to assign the "HMI Monitor" role to a user in addition to the "HMI Operator" role.



Note

Note that at least one user in the project has the "HMI Administrator" role and at least one user has access to the Control Panel. If access to the user management or the Control Panel is not possible, a complete download from the TIA Portal is necessary.

Adding users

Introduction

You can add a new user in the user list.

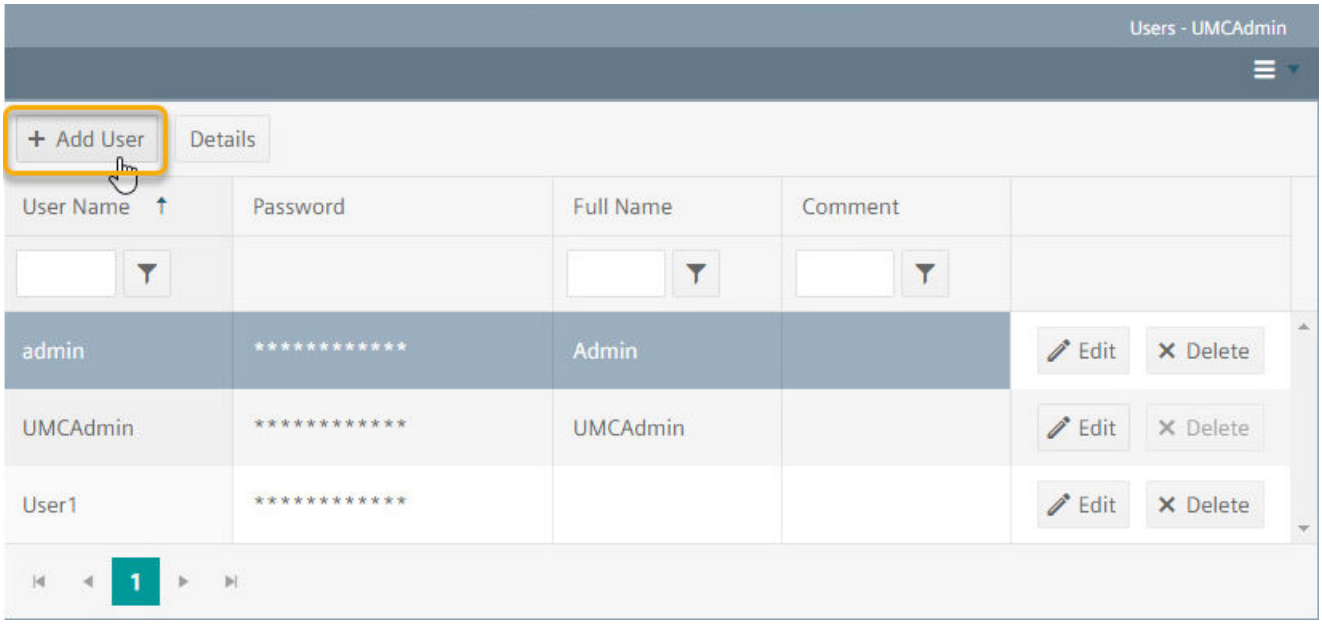
Requirement

- You have the "User management" function right.
- The home page of the user management is open.

Adding a new user

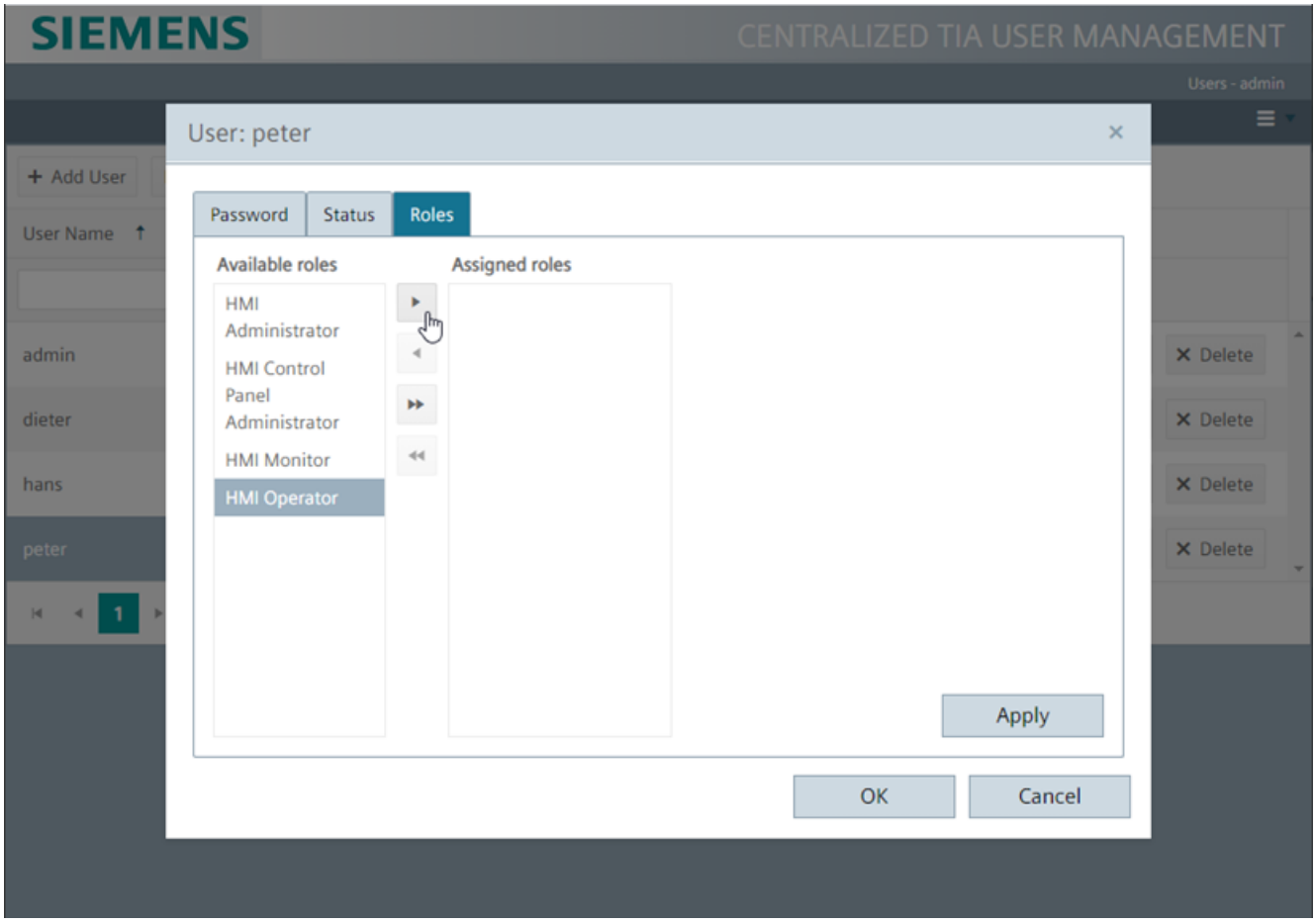
To add a new user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. In the user list, click "Add User".



3. A new row is displayed for the new user in the user list. Enter the information of the new user in the row.

4. Click "Details" in the user list. Assign roles to the new user.



5. Confirm your entries with the "Apply" button.
   Save the settings with "OK".

Deleting users

Introduction

You can delete a user in the user list.

Requirement

- You have the "User management" function right.
- The home page of the user management is open.

Deleting users

To delete a user, follow these steps:

1. Select "Users" in the menu. The user list is displayed.
2. Select the user.
3. Click the "Delete" button in the row. The user is deleted.

Deleting the user from the user list will become effective once the user logs off in Runtime.

Logging a user out

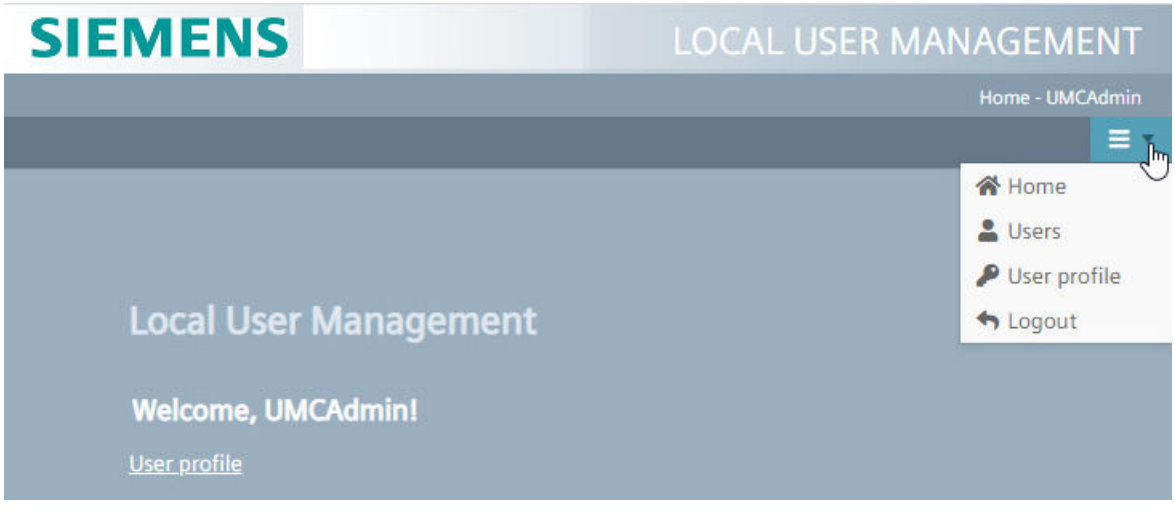
Introduction

You can log out from the user management.

Logging out

To log out, proceed as follows:

1. Close all open pages.
2. Select "Logout" from the menu.



You are logged out from runtime and from the user management.

Newly loaded data from the TIA Portal will not be applied until the next time you log in.

13.4.6 Using central user management

13.4.6.1 Setting central user management in the SIMATIC Runtime Manager

Multiple projects can be available on one PC. In SIMATIC Runtime Manager, you can activate the matching configuration of the user data for a selected project.

You cannot switch between local and central user management.

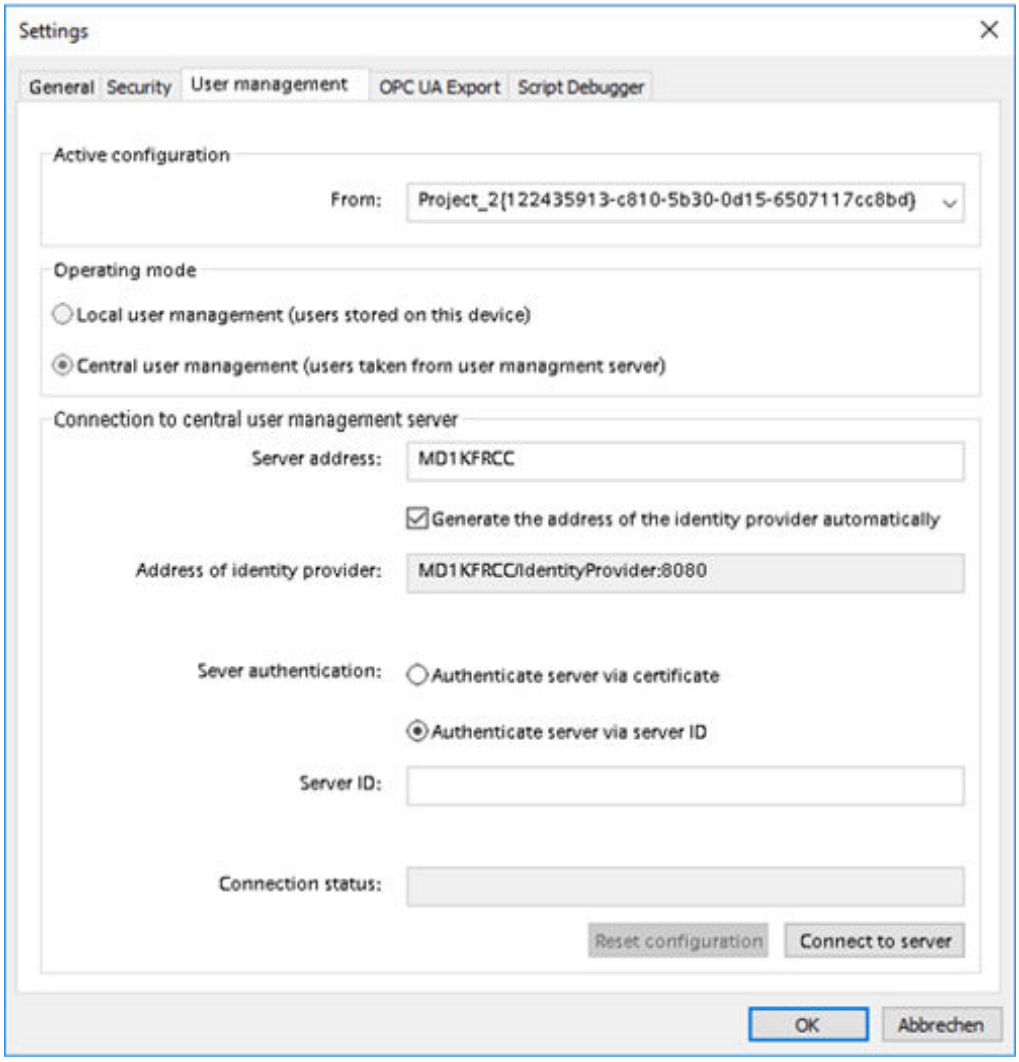
Requirement

- At least one project with user management is loaded on the Runtime PC.
- WinCC Unified Runtime is installed.
- SIMATIC Runtime Manager has been started.

Checking central user management in the SIMATIC Runtime Manager

When you have entered all information in the engineering system and the connection to the central user management can be established, all information is write-protected.

1. In the SIMATIC Runtime Manager, click "Settings" on the home page.
2. Select the "User management" tab.



3. When multiple projects are available on the PC, select the configuration of the user management of a project under "Select configuration".
4. Check the information in the area "Connection to UMC server".
5. Click the "Connect to server" button.
6. If the connection was successfully established, the status of the connection changes to "Connected" and the "Connect to server" button changes to "Check connection". You can reset the connection via "Reset configuration".

Configuring central user management

When the connection to the central user management could not be established, you can complete the information in the SIMATIC Runtime Manager.

1. In WinCC Unified Runtime, click "Settings" on the start page.
2. Select the "User management" tab.
3. When multiple projects are available on the PC, select the configuration of the user management of a project under "Activate configuration".
4. Complete the information. The setting "Authenticate server using server ID" is not changeable.
5. Click the "Connect to server" button.
6. If the connection was successfully established, the status of the connection changes to "Connected" and the "Connect to server" button changes to "Check connection". You can reset the connection via "Reset configuration".

See also

Managing multiple projects in the SIMATIC Runtime Manager (Page 6946)

13.4.6.2 Simulating a central user management

You want to simulate a project in which a central user management is configured for a customer. You have two options if you do not have access to the central user management of the customer:

- Configure your own central user management.
- Configure a local user management.

Requirement

- You know which groups and their function rights are contained in the central user management of the customer.

Configuring a central user management

Configure a central user management for the simulation project.

1. Create the users.
2. Create the user groups according to the customer project.
3. Assign the users to the groups.
4. Establish the connection to the central user management.
5. Start the simulation.
6. Log on in runtime.

Changes can be downloaded.

Configuring a local user management

Configure a local user management.

1. Create one or more users.
2. Assign the roles to the users.
3. Start the simulation.

Changes cannot be downloaded.

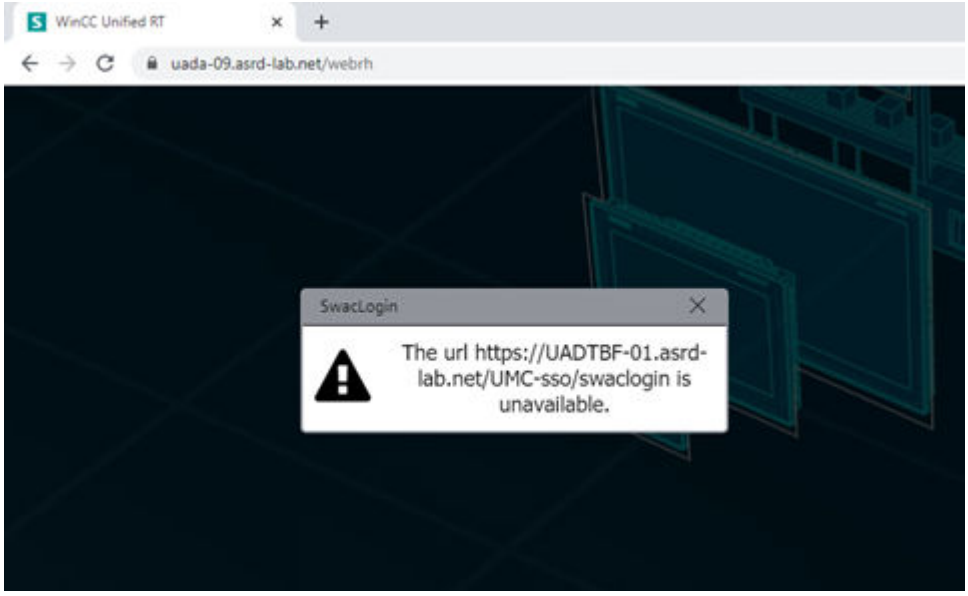
13.4.6.3 SwacLogin: Errors after complete download

After complete download of a project to a Unified PC, an error can occur when you open the WinCC Unified home page. The error can occur regardless of whether you open the home page locally on the PC or from a different device.

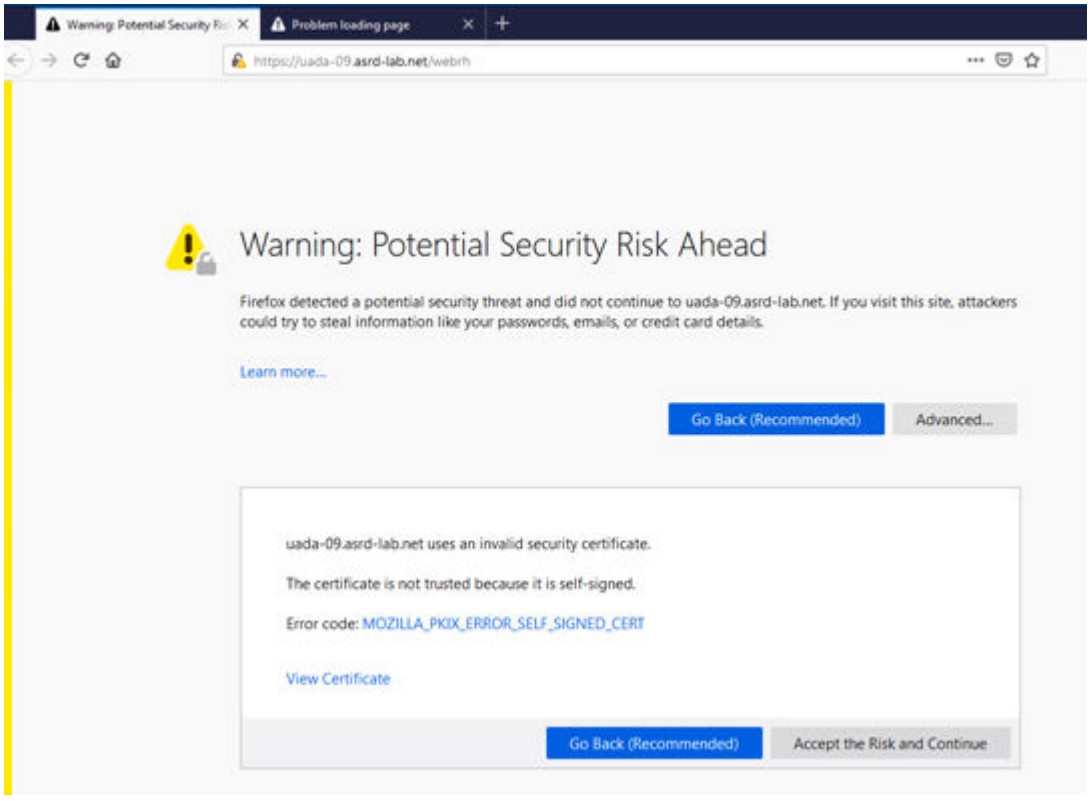
A possible cause of the error is the deletion of the browser cache.

Error description

In "Chrome" and "MS Edge", the error is displayed with the following alarm:



In "Firefox", the error is displayed with the following alarm:



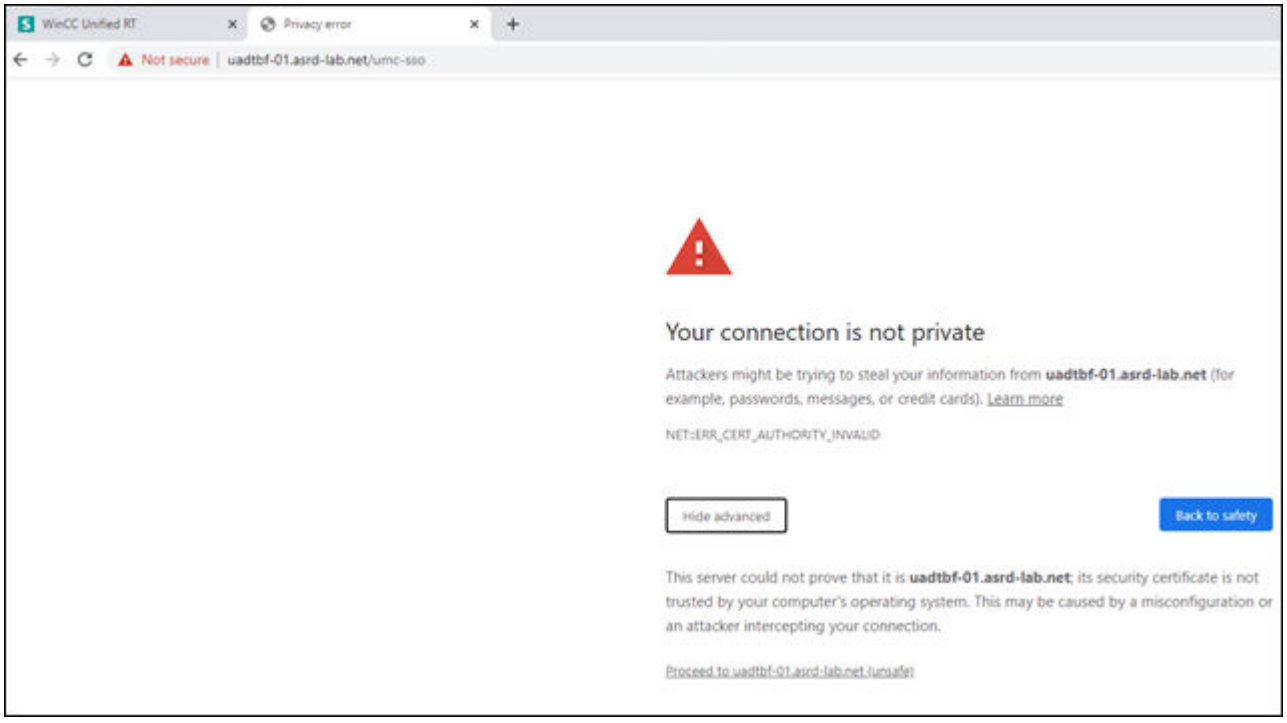
After accepting the warning of a potential security risk, the page remains empty in Firefox. Only the background screen is visible.

Remedy the error in "Chrome" and "MS Edge"

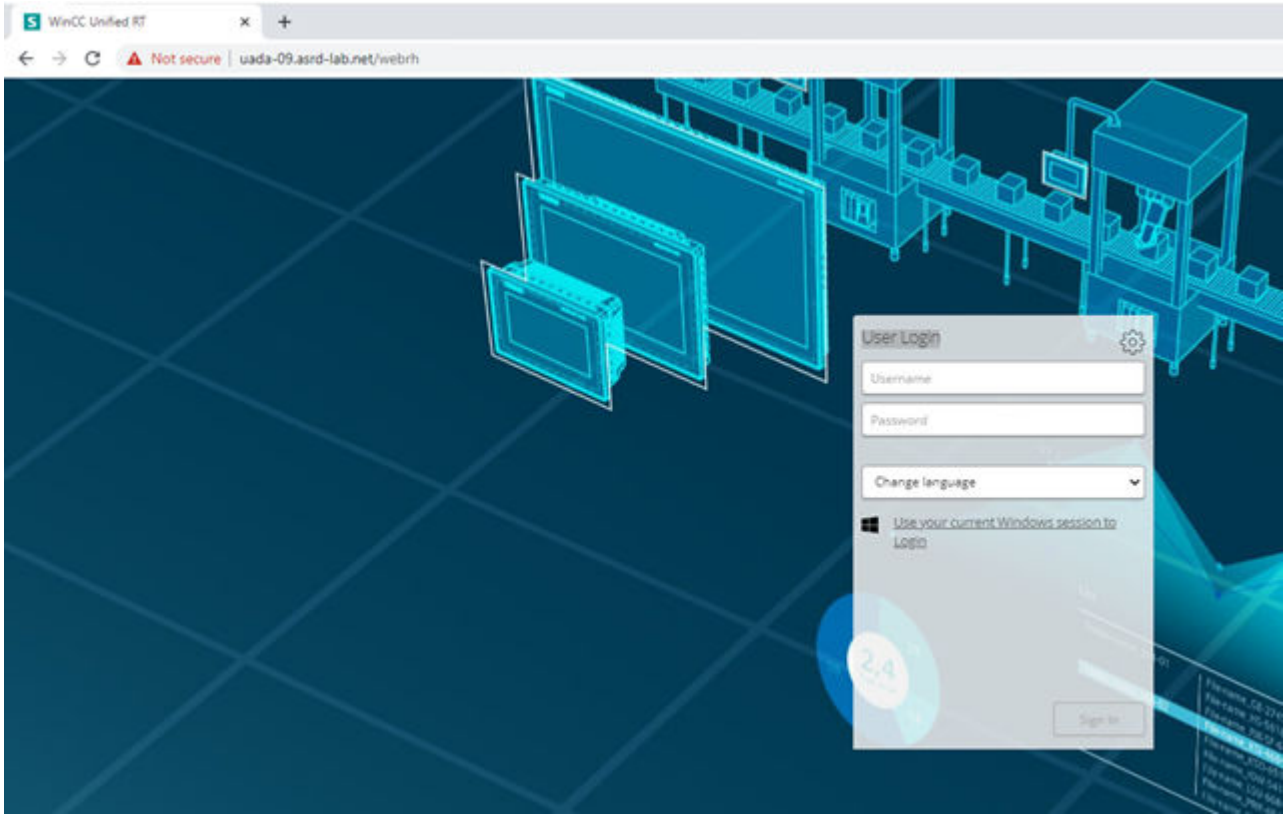
To fix the error in "Chrome" and "MS Edge", proceed as follows:

1. Open a new tab.
2. Enter the URL address of the identity provider of the UMC server in the address line of the browser. The URL is the same as the one in the error message without "/swaclogin", for example, "https://uadtbf-01.asrd-lab.net/umc-sso".

3. The page with a warning regarding the secure connection is displayed.



4. Accept the warning by clicking on "Proceed to uadtbf-01.asrd-lab.net (unsafe)".
5. The home page with the "User login" dialog is displayed.



Remedy the error in "Firefox"

To remedy the error in "Firefox", follow these steps:

1. Open a new tab.
2. Enter the URL address of the identity provider of the UMC server (ring server) in the address line of the browser, for example, "https://uadtbf-01.asrd-lab.net/umc-sso".
3. A blank page opens. Close the page.
4. Refresh the home page with the function key <F5>. The home page with the "User login" dialog is displayed.

Connectivity

14

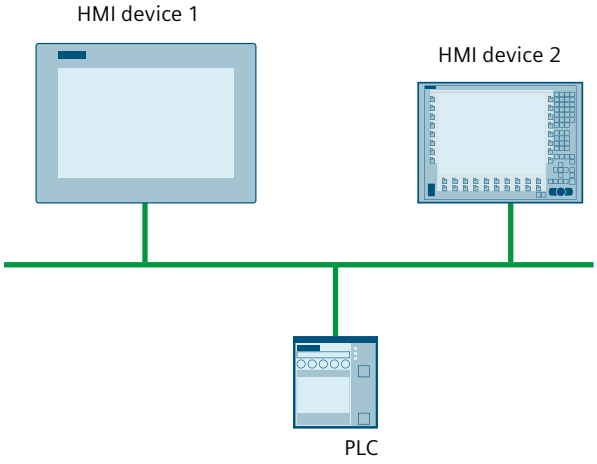
14.1 Basics

14.1.1 Basics of communication

14.1.1.1 Communication between devices

Communication

The data exchange between two devices is known as communication. The devices can be interconnected directly or via a network. The networked devices in communication are referred to as communication partners.



Data transferred between the communication partners is used for various purposes:

- Display processes
- Operate processes
- Output alarms
- Archive process values and alarms
- Document process values and alarms
- Administer process parameters and machine parameters

Communication partners in the automation system

An automation system consists of the following communication partners:

- PLC
  The PLC controls a process by means of a user program.
- HMI device
  The HMI device is used to operate and monitor the process.
  Communication between the communication partners PLC and HMI device is described below.
  Additional information on other forms of communication is available in the online help of the TIA Portal in the section "Editing devices & networks".
  If the following requirements are met, the PLC and HMI device form an automation system:
  - PLC and HMI device are linked to each other
  - Network between PLC and HMI device is configured

Network configuration

The basis for all types of communication is a network configuration.

- Every device in a network has a unique address.
- The devices carry out communication with consistent transmission characteristics.

Data exchange using tags

Process values such as temperatures and levels are transferred by tags in Runtime. Process values are stored in the memory of one of the connected automation systems.

To access the process data with the HMI device, link the external HMI tags to the PLC tags.

For additional information on configuring tags, refer to "Configuring tags (Page 607)".

Communication via a uniform and vendor-neutral interface

With OPC UA (Open Platform Communications Unified Architecture), WinCC has a uniform and manufacturer-independent software interface. This interface enables standardized data exchange between industrial, office, and manufacturing applications.

For more detailed information, refer to the documentation for OPC UA.

See also

Supported PLCs and communication channels (Page 6981)

Configuring communication (Page 6975)

14.1.1.2 Configuring communication

Introduction

To configure an automation system, you configure the connections in the "Devices & networks" editor. Connections of devices that are within the same project and were created with the "Devices & networks" editor are referred to as integrated connections. Connections of devices that were created with the "Connections" editor are referred to as non-integrated connections. The devices do not all have to be within the same project.

You use the graphic and table network view for the configuration.

Requirement

- The network configuration is complete.
- HMI device and PLC are available in the hardware catalog.

Note

Non-integrated configuration

If integrated configuration of the HMI connections is not possible, create non-integrated HMI connections of the HMI device using the "Connections" editor.

Procedure

To set up an automation system, always follow the steps below:

1. **Inserting devices**
   You drag a PLC and an HMI device from the hardware catalog to the network view of the "Devices & networks" editor.
2. **Configuring devices**
   Depending on the HMI device used, you add the required communications modules to your PC station.
3. **Networking devices**
   In the networking step, you configure the physical connection of the devices.
   To connect the devices, you connect the interfaces of the devices with communications capability using drag and drop.
4. **Connecting devices**
   To set up a logical communication connection between the communication partners, you create an HMI connection between the networked devices.
   The tabular network overview supplements the graphical network view.
   In addition, the created HMI connection is also visible in the "Connections" editor of the HMI device where it can be configured.

See also

Supported PLCs and communication channels (Page 6981)

Networking the HMI device and PLCs (Page 6985)

Creating an integrated HMI connection (Page 6988)

Creating a non-integrated HMI connection (Page 6992)

Communication between devices (Page 6973)

14.1.1.3 Secure communication and certificates

Runtime Unified supports secure communication to PLCs of the S7 series, for example, S7-1200 and S7-1500.

Each connection is encrypted by the TLS protocol.

In the TIA Portal you configure secure communication to the PLCs with a PLC certificate.

Note

Encryption with TLS

Always use the most current version of TLS. Disable the older version.

The use of older versions (TLS 1.0 und 1.1) is at your own risk.

Also observe the instructions in SiePortal:

- Making use of Runtime by using certificates (https://support.industry.siemens.com/cs/ww/en/view/109806850)
- Security guidelines for SIMATIC HMI operator devices and SIMATIC WinCC Unified (https://support.industry.siemens.com/cs/ww/en/view/109481300)

Connection establishment

Establishing an unencrypted connection

The connection establishment to PLCs, such as S7-1200 and S7-1500, for example, with older firmware or without PLC certificates is still possible.

Establishing an encrypted connection with secure communication

Current firmware allows for a connection with secure communication via the TLS protocol.

Certificates of a certificate authority, end unit certificates (self-signed or trusted) and certificate revocation lists, if available, are transferred to create an encrypted session.

The session is set up when the certificates match. If the certificates do not match, setup of the session is rejected, and a system alarm is generated.

Creating a connection with secure communication from Runtime Unified to the PLC

The certificates for Runtime Unified are automatically handled by the engineering system and sent to the device.

However, if certificates are modified in the PLC and the HMI device is not reloaded, the following scenarios can be created.

Connection of the Runtime Unified without certificates to a PLC without certificate

- The connection is established during the next connection.

Connection of the Runtime Unified to a PLC with the same certificate

- The connection is established during the next connection.

Connection of the Runtime Unified without certificates to a PLC with certificate

- When the PLC is configured so that HMI connections with and without secure communication are permitted, the connection is established.
- If the PLC only permits connections with secure communication, the connection is denied. The PLC certificate is saved in "%PROGRAMDATA%/Siemens/Automation/device-certificate-store/**untrusted**/certs" . The certificate file contains the IP address of the PLC and a fingerprint of the certificate.
- The HMI device tries every 30 seconds to set up the connection again. The PLC accepts the certificate during the next attempt and the connection is established.

Connection of the Runtime Unified to a PLC with different certificates

- If the HMI connection has a certificate, an attempt is made to set up a connection to this certificate. If the certificate of the HMI connection does not match the one of the PLC, the connection is denied. The PLC certificate is saved in "%PROGRAMDATA%/Siemens/Automation/device-certificate-store/**untrusted**/certs" . The certificate file contains the IP address of the PLC and a fingerprint of the certificate.
- The HMI device tries every 30 seconds to set up the connection again. The PLC accepts the certificate during the next attempt and the connection is established.

Certificate revocation list (CRL)

A certificate revocation list contains certificates that were revoked before they expired and, therefore, are no longer trusted. A certificate revocation list is provided as a CERT file (encoded in PEM or DER format). More than one certificate revocation list can exist.

Certificate revocation lists are saved in the path

"%PROGRAMDATA%/Siemens//Automation//device-certificate-store//trusted//crl/"

.

While the HMI connection to a PLC is being established, the PLC certificate is compared with the certificate revocation list. If the certificate is listed there, the connection is denied. An alarm is output.

14.1.1.4 Networks and connections

SIMATIC communication networks

Communication networks

Overview

Communication networks are a central component of an automation solutions. Industrial networks fulfill special requirements:

- Coupling of automation systems as well as simple sensors, actuators, and PCs
- Error-free transfer of information at the right time
- Robustness against electromagnetic interference, mechanical stresses and soiling
- Flexible adaptation to the production requirements

Industrial networks belong to the LANs (Local Area Networks) and allow communication within a limited area.

Industrial networks fulfill the following communication functions:

- Process and field communication of the automation systems including sensors and actuators
- Data communication between automation systems
- IT communication for integrating information technology

HMI devices in the plant network

You connect an HMI device in the network to SIMATIC S7 modules that have an integrated interface of the corresponding communication channel.

You can connect multiple HMI devices to one SIMATIC S7 PLC and multiple SIMATIC S7 PLCs to one HMI device. The maximum number of communication partners that you can connect to an HMI device is dependent on the HMI device used.

Additional information is available in the documentation for the respective HMI device.

See also

PROFINET Industrial Ethernet (Page 6978)

PROFINET Industrial Ethernet

PROFINET

PROFINET is an open standard for industrial automation defined by IEEE 61158 and based on Industrial Ethernet. PROFINET makes use of IT standards all the way to the field level and enables plant-wide engineering.

With PROFINET, you realize automation solutions, the high performance and communication in real-time requirements.

Industrial Ethernet

Industrial Ethernet, which is based on IEEE 802.3, enables you to connect your automation system to your office networks. Industrial Ethernet provides IT services that you can use to access production data from the office environment.

See also

Communication networks (Page 6978)

Connections

HMI connection

Definition

An HMI connection is a logical connection between an HMI device and a PLC. The HMI connection enables communication between the communication partners.

Unlike an S7 connection, the HMI connection is assigned to the HMI device.

Layout

The HMI connection defines the following within the plant network:

- Communication partners
  The HMI connection identifies the devices in the plant configuration.
- Communication channel over which these communication partners communicate.
  The HMI connection requires a configured network.
- Communication path
  The HMI connection defines the interface parameters and the network addresses of the communication partners.

HMI connection types

The options for addressing external tags depend on the type of HMI connection between WinCC and the PLC in question. The TIA Portal supports the following types of connection:

- Integrated HMI connection
  In the TIA Portal you configure integrated HMI connections between the devices in the "Devices & Networks" editor. An integrated HMI connection enables an optimized data exchange.
- Non-integrated HMI connection
  In the case of a non-integrated connection, the PLC program can be created outside the WinCC project. You configure the PLC and the WinCC project independently each other. For configuration in WinCC, you only need to know the addresses used in the PLC and their function.
  You use a non-integrated HMI connection, for example, in the following application cases:
  - You configure a WinCC project for external PLCs.
  - You do not have access to the device configuration of a SIMATIC PLC, for example, because you are working without a STEP 7 license.

  You configure a non-integrated HMI connection for the HMI device in the "Connections" editor of the WinCC project.

See also

Creating an integrated HMI connection (Page 6988)

Creating a non-integrated HMI connection (Page 6992)

Additional connection types (Page 6980)

Setting up switch on/switch off of a connection in runtime (Page 6993)

Additional connection types

Overview

The following table provides an overview of the connection types that you can use in addition to the HMI connection for communication to specific device types and areas of application.

Additional information on connection types is available in the online help of the TIA Portal in the section "Editing devices & networks".

Connection type	Description	Application
S7 connections	Connection type can be used in all S7 devices	Data exchange between SIMATIC S7 stations
FDL connection	Fieldbus Data Link Security layer Based on PROFIBUS	Communication with a partner that supports sending and receiving according to the SDA function (Send Data with Acknowledge), e.g. SIMATIC S5 or PC.

Connection type	Description	Application
ISO transport connection	Suitable for large amounts of data Based on ISO transport	Communication with a partner that supports sending and receiving data in accordance with ISO transport, e.g. SIMATIC S5 or PC.
ISO-on-TCP connection	Transmission Control Protocol/Internet Protocol with the extension RFC 1006 Corresponds to the standard TCP/IP	Communication with a partner that supports sending and receiving of data in accordance with ISO-on-TCP, e.g. PC or external system.
TCP connection	Transmission Control Protocol/Internet Protocol Corresponds to the standard TCP/IP	Communication with a partner that supports sending and receiving data in accordance with TCP/IP, e.g. PC or external system.
UDP connection	User Datagram Protocol Subnet: Industrial Ethernet	Unsecured transmission of related data fields between two nodes
Email connection	In the case of an email connection, the mail server via which all emails sent by an IT-CP are delivered is defined.	For example, enables the sending of process data, for example, from data blocks via email using a CP with IT functionality (IT-CP);
P2P connection	Peer-to-Peer Communication between two equal devices	Communication with external devices. e.g. a printer.

See also

HMI connection (Page 6979)

Supported PLCs and communication channels

Overview

Your HMI device can communicate with the following SIMATIC PLC families via integrated HMI connections:

SIMATIC PLC family	Supported communication channels	Comment
SIMATIC S7-1200/1500	Industrial Ethernet	Parallel communication with several PLCs is possible
SIMATIC S7-300/400	Industrial Ethernet	Parallel communication with several PLCs is possible

Communication drivers

In the case of non-integrated connections, the HMI devices, PC systems and PLCs communicate via the following communication drivers:

Communication drivers	Interface/Communication channel	HMI device/ Panel	PC system	Comment
SIMATIC S7-1200/1500	Industrial Ethernet/PROFINET	x		Parallel communication with several PLCs is possible
	Ethernet MPI/DP		x	
SIMATIC S7-300/400	Industrial Ethernet/PROFINET	x		Parallel communication with several PLCs is possible
	Ethernet MPI/DP		x	
SIMATIC HMI HTTP	Ethernet ⇒ http/https		x	
Allen-Bradley Ethernet IP	Industrial Ethernet	x		Parallel communication with several PLCs is possible
Allen Bradley DF1	COM interface		x	
Mitsubishi FX	COM interface		x	Parallel communication with several PLCs is possible
Mitsubishi iQr/iQF	Industrial Ethernet	x		
Mitsubishi MC TCP/IP	Industrial Ethernet	x		Parallel communication with several PLCs is possible
	Ethernet		x	
Modbus RTU	COM interface		x	
Modbus TCP/IP	Industrial Ethernet	x		Parallel communication with several PLCs is possible
	Ethernet		x	
Omron Ethernet/IP	Industrial Ethernet	x		Parallel communication with several PLCs is possible
Omron Host Link	COM interface		x	
OPC UA	OPC	x	x	Parallel communication via OPC UA connections possible
LOGO!	Ethernet		x	

See also

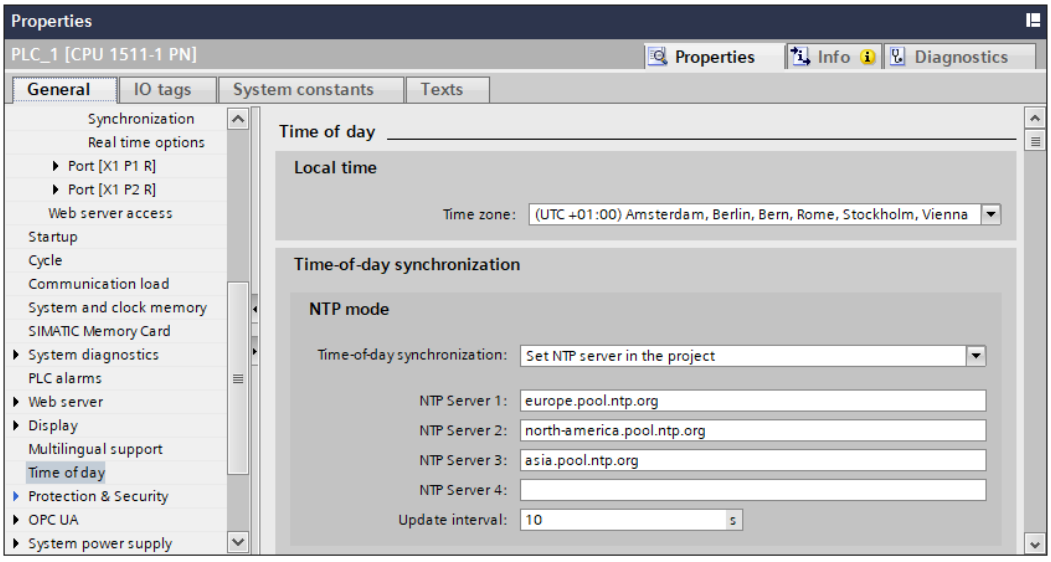
Communication between devices (Page 6973)

Configuring communication (Page 6975)

14.1.1.5 Synchronization

Time synchronization on the S7-1200/1500

1. To make settings for the time synchronization, select the "Online & Diagnostics" node of the PLC in the project tree.
2. In the Inspector window, select "Properties > General > Time of day".



Note

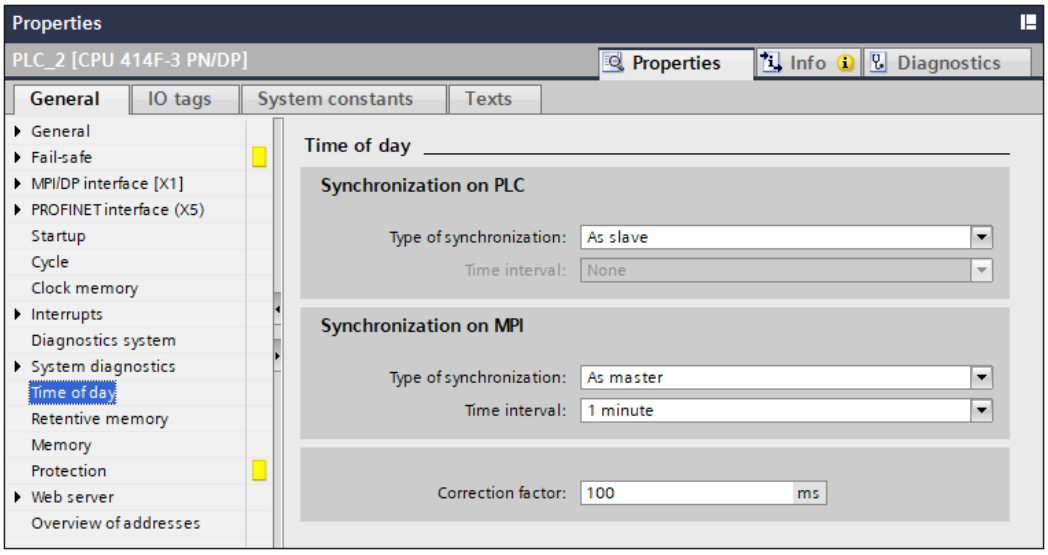
Example of window

Different settings are available depending on the configured PLC.

3. Select the time zone where the device is located.
4. Enable the time synchronization for the device by selecting how the NTP server is accessed under "Time synchronization".
5. Specify at least one NTP server.
6. Complete the settings for
   - Update interval
   - Daylight saving time
   - Start of standard time

Time synchronization on the S7-300/400

1. To make settings for the time synchronization, select the "Online & Diagnostics" node of the PLC in the project tree.
2. In the Inspector window, select "Properties > General > Time of day".



Note

Example of window

Different settings are available depending on the configured PLC.

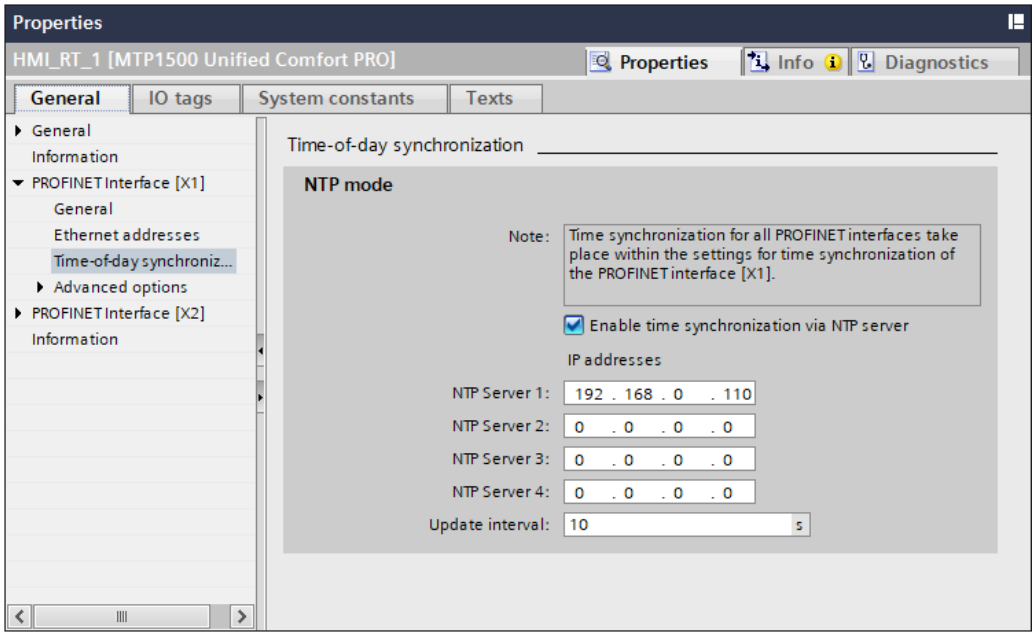
3. Select the synchronization type for the interfaces.
4. Specify a correction factor.

Time synchronization for PROFINET interfaces on Panels

Note

Activating this service reduces security against unauthorized access to functions and data of the PLC from the outside and via the network.

1. To make settings for the time synchronization, select the "Online & Diagnostics" node of the panel in the project tree.
2. Select "Properties > PROFINET interface [x] > Time synchronization" in the Inspector window.



3. Enable the time synchronization.
4. Specify at least one NTP server.

14.1.2 Configuring an HMI connection

14.1.2.1 Configuring an integrated HMI connection

Networking the HMI device and PLCs

Introduction

You can network an HMI device to several PLCs. The networking of devices is depicted by lines that are colored depending on the interface type.

The number of available interfaces and interface types depends on the device. To make additional interfaces available on the device, add a communications module to the device.

Requirement

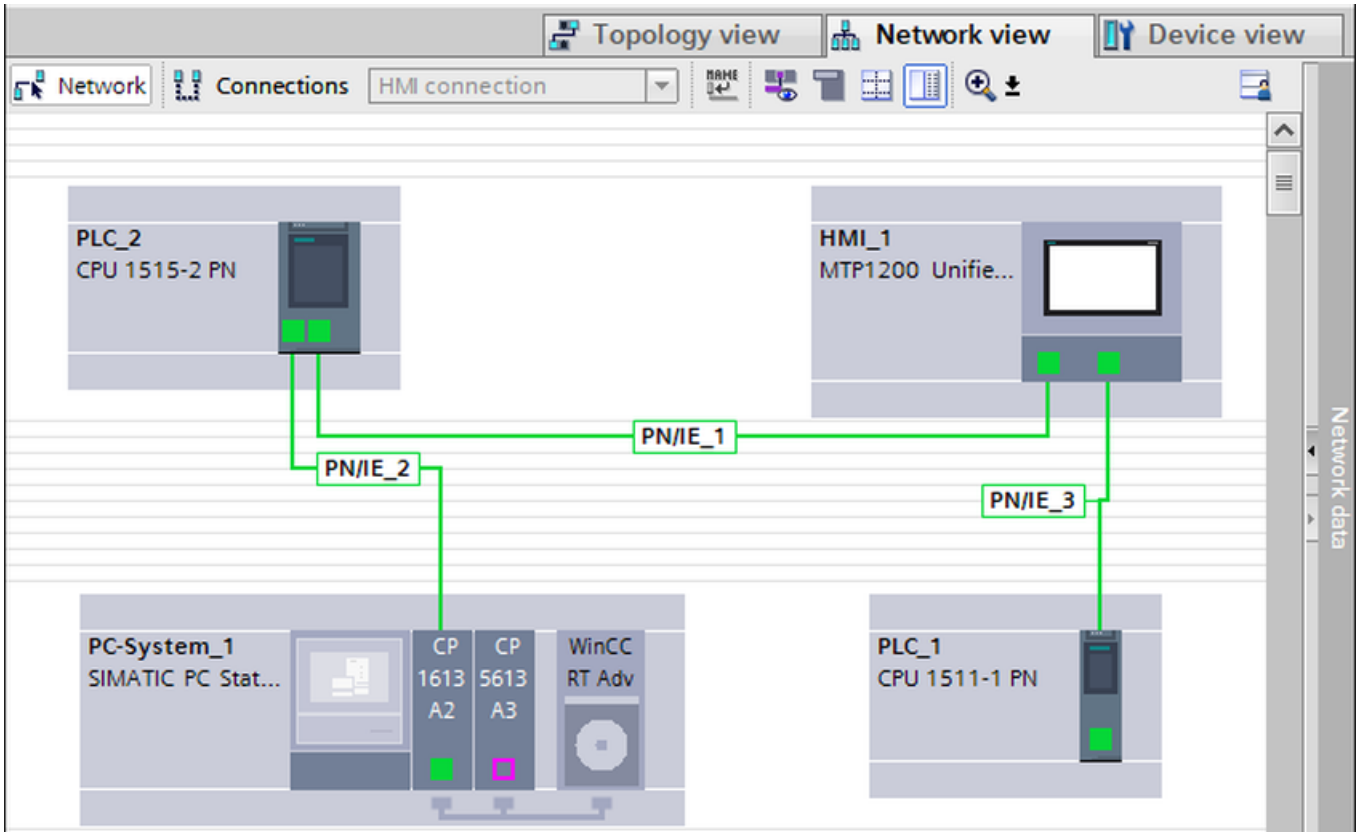
- The "Devices & Networks" editor is open.
- The networks are configured.

- An HMI device is configured in the "Devices & Networks" editor.
- The PLC is configured in the "Devices & Networks" editor.

Procedure

To network an HMI device and a PLC, follow these steps:

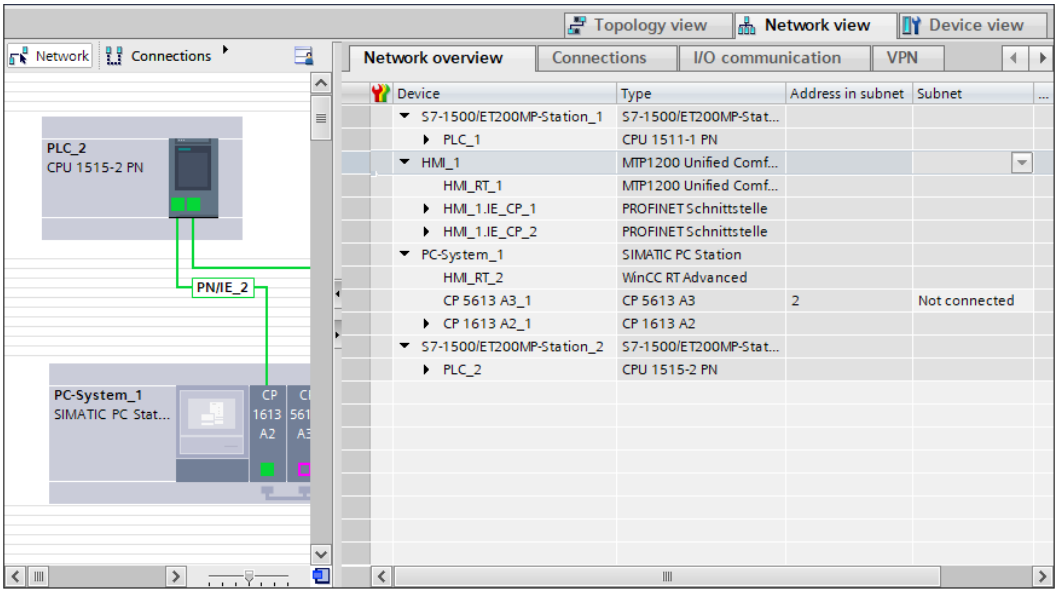
1. Open the network view of the "Devices & Networks" editor.
2. Enable the "Networking" mode.
3. Use a drag-and-drop operation to interconnect the interfaces of the desired communication network of the devices.
   A connection is shown as graphic and table in the network view.



The tabular network overview supplements the graphical network view with the following additional functions:

– You obtain detailed information on the structure and parameter settings of the devices.

– Using the "Subnet" column, you can connect communication-capable components to subnets that have been created.



See also

Creating an integrated HMI connection (Page 6988)

SIMATIC communication networks (Page 6978)

Creating an integrated HMI connection

Introduction

An integrated HMI connection connects an HMI device and a SIMATIC S7 PLC within your project.

Connection resources

Each connection requires connection resources for the end point or transition point on the devices involved. The number of connection resources is device-specific.

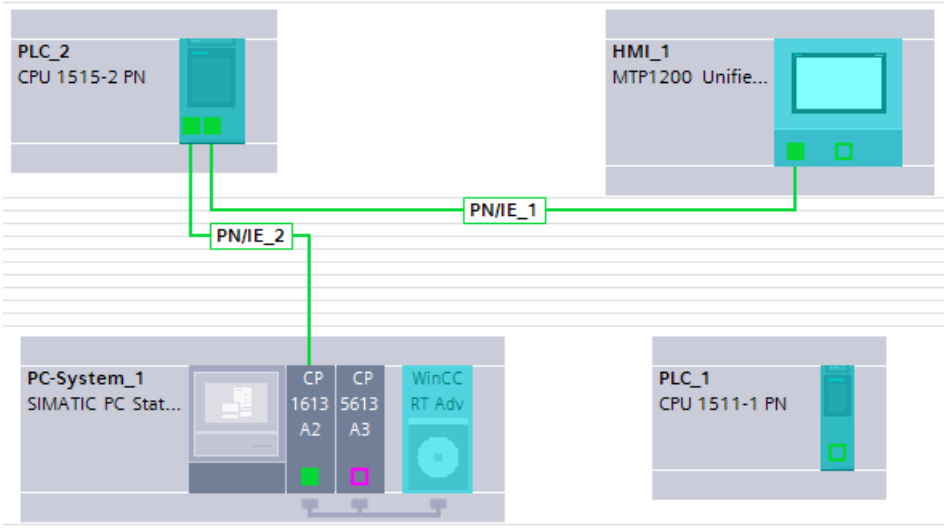
If all connection resources of a communication partner are allocated, no new connection can be configured.

Requirement

- The networks are configured.
- An HMI device and a SIMATIC PLC are configured and networked.
- The network view is open in the "Devices & Networks" editor.

Create an integrated HMI connection

1. Enable the "Connections" mode.
2. Select the "HMI connection" connection type.
   The devices available for connection are highlighted in color.



3. Use a drag-and-drop operation to interconnect the interfaces of the desired communication channel of the HMI device and PLC with each other.
   The HMI connection is shown as graphic and table in the network view.
   In the table area of the editor, the HMI connection is displayed on the "Connections" tab.

Note

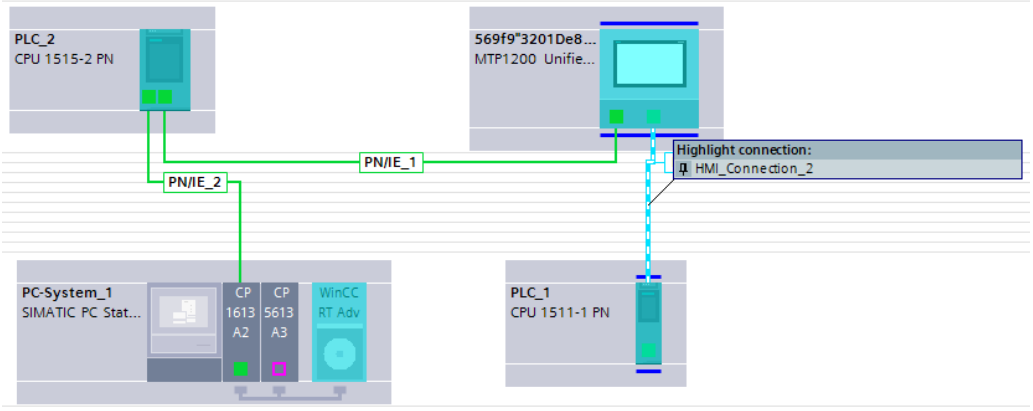
Change local connection names

You can change the local name for the connection only in the tabular area of the editor.

4. Change the connection parameters in the tabular area according to the requirements of your project.

Open the graphic view of the connection partners

1. Select the HMI connection.
2. Click "Highlight HMI connection" and select the HMI connection.



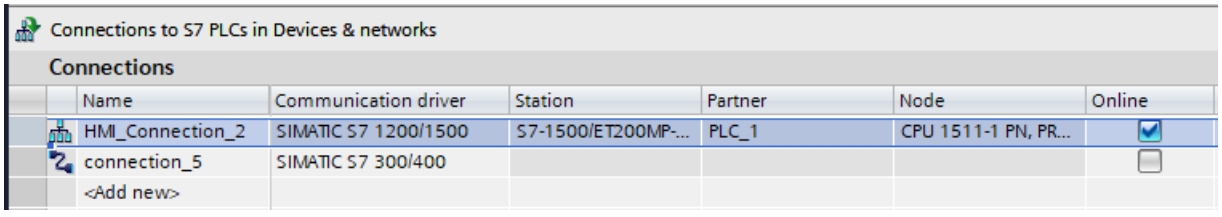
The connection path is shown in the Inspector window under "Properties > General > General".

Change the connection path

1. Open the graphic view display of the connection partners.
2. Select a different interface in the Inspector window under "Properties > General > General > Interface".
   The existing connection parameters are highlighted as invalid.
3. To validate the connection parameters, click on "Find connection path".
   The connection parameters are reassigned and validated.

Create an integrated HMI connection in the "Tags" editor

1. Double-click on "HMI tags" below your HMI device in the project tree.
2. Select a tag table.
   The "HMI tags" editor opens.
3. Create an HMI tag.
4. Connect the HMI tag to an existing PLC tag of the matching data type.
5. The integrated HMI connection to the PLC is established automatically.



See also

HMI connection (Page 6979)

Networking the HMI device and PLCs (Page 6985)

S7-1500 | Integrated HMI connection (Page 6999)

S7-300/400 | Integrated HMI connection (Page 7004)

14.1.2.2 Configuring a non-integrated HMI connection

Configuring non-integrated connections

Introduction

A non-integrated HMI connection requires a communication driver and a good understanding of the address structure of the communication partner.

Addressing with non-integrated connections

In the case of a project with a non-integrated connection, you always configure a tag connection exclusively with absolute addressing.

Select the valid data type yourself. If the address of a PLC tag changes in a project with a non-integrated connection during the course of the project, you also have to perform the change in WinCC. The tag connection is not checked for validity in Runtime. No alarm is displayed.

Communication drivers

A communication driver is a software component that establishes a connection between a PLC and an HMI device. The communication driver thus enables the assignment of process values to HMI tags.

Depending on the HMI device used and the connected communication partner, you select the interface used as well as the profile and transmission speed.

Basic procedure

The following steps are required to work in your project in a non-integrated connection:

1. Create an HMI connection
2. Select communication drivers and interfaces
3. Address the communication partners
4. Assign the communication network
5. Close the connection

See also

Creating a non-integrated HMI connection (Page 6992)

Supported PLCs and communication channels (Page 6981)

Creating a non-integrated HMI connection

Introduction

A non-integrated connection connects an HMI device to a PLC that is configured outside your project. You create the non-integrated HMI connection in the "Connections" editor of the HMI device.

Requirements

- A project is open.
- An HMI device has been created.

Procedure

To create a non-integrated connection, follow these steps:

1. Double-click "Connections" in the project tree below your HMI device.
   The "Connections" editor opens.
   Existing integrated connections are identified with  .
   Existing non-integrated connections are identified with .
2. Create a new connection with "Add".
3. Select the communication driver. Use the communication driver of the required PLC family.
4. Select the required interface of the HMI device in the graphic area of the editor under "Parameters > [HMI device type] > Interface".
   The number of available interfaces on the HMI device depends on the communication driver.
5. Change the connection parameters according to the requirements of your project.

See also

HMI connection (Page 6979)

Configuring non-integrated connections (Page 6991)

Supported PLCs and communication channels (Page 6981)

S7-1500 | Non-integrated HMI connection (Page 7000)

S7-300/400 | Non-integrated HMI connection (Page 7005)

14.1.2.3 Setting up switch on/switch off of a connection in runtime

Introduction

If an HMI device and a PLC do not have to always be connected, terminate the connection in Runtime and establish it again when necessary. This reduces the load on the communication channel.

Configure a Runtime script for enabling/disabling a connection in runtime.

Note

Alarm system and system diagnostics

After switching off the connection to a PLC, the alarms from this PLC continue to be displayed. The system diagnostics for this PLC is also available.

Requirement

- An HMI connection is configured.
- A button is configured in the HMI device of the HMI connection.
- The "Screens" editor is open.

Procedure

To configure enabling/disabling of a connection in Runtime, follow these steps:

1. Select a button.
2. Select the event that is to trigger enabling/disabling in runtime under "Properties > Events" in the Inspector window.
3. Program a script to the event which enables or disables the connection via the "Set Connection mode" snippet.

Result

Pressing this button triggers enabling/disabling of the connection in Runtime.

See also

Introduction to runtime scripting (Page 969)

HMI connection (Page 6979)

14.1.3 Device configuration

14.1.3.1 HMI devices

Panels and PC systems are used as HMI devices.

Definition

An HMI device visualizes the plant process, shows the process values and enables access to the plant control system via operator inputs.

The HMI device needs a runtime software for process visualization and operation. The device must have the corresponding interfaces and communication drivers to connect the HMI device to the plant network and the PLC.

Structure in the device navigation

Edit further components, to configure a PC system:

- PC station
  The hardware basis of a PC system is an industrial PC.
  The PC provides the operating system, the firmware and the hardware equipment.
- WinCC Runtime software
  The Runtime software visualizes your WinCC Runtime project and enables process operation.
  The runtime software is available in the hardware catalog.
- Communications modules
  If the PC station does not have the required interfaces, install the required communications modules in the PC station.

A Panel is a complete integrated HMI device which does not require further components.

See also

Inserting a HMI device into the project (Page 6994)

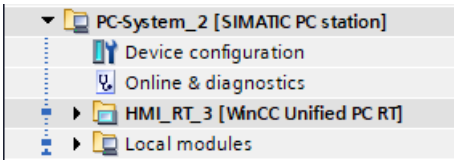
14.1.3.2 Inserting a HMI device into the project

Requirements

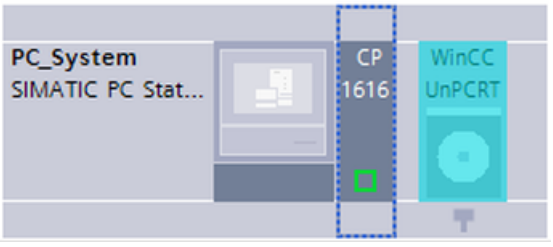
- The networks are configured.
- HMI device and PLC each support the communication channel of the respective network.
- The network view is open in the "Devices & Networks" editor.
- A corresponding interface is required at both ends to connect the HMI device to the PLC.

Configuring a SIMATIC WinCC Unified PC

1. Drag the SIMATIC WinCC Unified PC from the hardware catalog to the work area.
   A SIMATIC PC system with the WinCC Unified PC RT is created.



2. Select a communication module for the required interface type in the hardware catalog.
3. Drag the communication module onto the SIMATIC WinCC Unified PC.
   The communication module is added.



4. Drag a PLC from the hardware catalog to the work area.
   The PLC is created.
5. Network the devices.

Configuring a SIMATIC Unified Comfort Panel

1. Drag a SIMATIC Unified Comfort Panel from the hardware catalog to the work area.
   The SIMATIC Unified Comfort Panel is created.
2. Drag the matching PLC from the hardware catalog to the work area.
   The PLC is created.
3. Network the devices.

See also

HMI devices (Page 6994)

14.2 Communication with SIMATIC PLCs

14.2.1 Communicating with SIMATIC S7-1200/1500

14.2.1.1 Communication with S7-1200/1500

Overview

You can configure the following communication channel for communication between an HMI device and the SIMATIC S7-1200/1500 PLC.

- PROFINET

See also

Permitted data types for SIMATIC S7-1200/1500 (Page 6996)

Symbolic addressing (Page 6997)

14.2.1.2 Permitted data types for SIMATIC S7-1200/1500

Permitted data types for connections with SIMATIC S7-1500

The table below lists the data types that can be used when configuring tags.

Data type	Length
BOOL	1 bit
BYTE	1 byte
CHAR	1 byte
DATE	2 bytes
DATE_AND_TIME	8 bytes
DINT	4 bytes
DTL	12 bytes
DWORD	4 bytes
INT	2 bytes
LDT	8 bytes The value range of the data type LDT of an S7-1500 stretches from 1970-1-1 00:00:00.000000000 to 2262-04-11 23:47:16.854775807.
LINT	8 bytes
LREAL	8 bytes
LTIME	8 bytes
LTIME_OF_DAY	8 bytes
REAL	4 bytes

Data type	Length
S5TIME	2 bytes
SINT	1 byte
STRING	(2+n) bytes, n = 0 to 254
TIME	4 bytes
TIME_OF_DAY	4 bytes
UDINT	4 bytes
UINT	2 bytes
ULINT	8 bytes
USINT	1 byte
WCHAR	2 bytes
WORD	2 bytes
WSTRING	(2+n) 2 bytes, n = 0 to 254

See also

Communication with S7-1200/1500 (Page 6996)

Symbolic addressing (Page 6997)

14.2.1.3 Symbolic addressing

Introduction

Data are exchanged between the HMI device and the PLC via tags.

Depending on the addressed data blocks, these tags are addressed as absolute or symbolic in the PLC.

- Symbolic addressing
  For symbolic addressing, a validity check of the tag connection is performed in runtime. If an address is changed in the PLC, the change is registered and an error message is issued.
  For symbolic addressing, select the PLC tag via its name and connect it to an external HMI tag. The valid data type for the external HMI tag is automatically selected by the system.
- Absolute addressing
  The linking of tags is not checked in runtime. You select the valid data type of the tags. If the tag address changes in the PLC, compile and load the HMI device again so that the change is registered in Runtime.

Data blocks and symbolic access

Data blocks with optimized access support only symbolic addressing.

During the symbolic addressing of a data block, the address of an element in the data block is dynamically assigned and is automatically adopted in the HMI tag in the event of a change.

Neither the connected data block nor the WinCC project must be compiled.

For symbolic addressing of elements in a data block, you need to recompile and reload the WinCC project only in case of the following changes:

- Name or data type of the connected data block element or of the global PLC tag
- Name or data type of a higher-level structure node of the connected element in the data block element or global PLC tag
- Number of the connected data block

HMI connections and symbolic access

With symbolic addressing of tags, you create an integrated HMI connection:

- Integrated connection
  You address the tags symbolically as well as absolutely over an integrated connection.
- Non-integrated connection
  You address the tags only absolutely over a non-integrated connection.
  A non-integrated connection is available for all supported PLCs.

Disabling symbolic access

In WinCC, symbolic addressing is the default method.

To change the default setting, follow these steps:

1. Select "Options > Settings > Visualization > HMI tags" in the menu.
2. Activate the "Symbolic access" option.

See also

Communication with S7-1200/1500 (Page 6996)

Permitted data types for SIMATIC S7-1200/1500 (Page 6996)

14.2.1.4 Interface and communication parameters

S7-1500 | Integrated HMI connection

PROFINET interface parameters

The following table shows the interface parameters of an integrated HMI connection:

Table 14-1 PROFINET parameters of the HMI device

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
MAC address	Specifies the MAC address for the connection type "ISO connection". This option is only available when the "Use ISO protocol" option is selected.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.

Table 14-2 PROFINET parameters of the PLC

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.
PROFINET device name	Shows the PROFINET device name or specifies it. This property is only available when the "Generate PROFINET device name automatically" option is deactivated.
Converted name	Shows the name that is automatically generated from the PROFINET device name and satisfies the DNS conventions.
Device number	Shows the device number by which an IO device can be identified.

S7-1500 | Non-integrated HMI connection

PROFINET interface parameters

The following table shows the interface parameters of a non-integrated HMI connection:

Table 14-3 PROFINET parameters of the HMI device

Parameters	Description
Interface	Specifies the communication channel.
Address	Specifies the IP address of the device.
Access point	Defines a logical device name through which the communication partner can be reached.

Table 14-4 PROFINET parameters of the PLC

Parameters	Description
Address	Specifies the IP address of the device.

14.2.1.5 Troubleshooting for SIMATIC S7-1200/1500

Causes of a faulty connection

Causes for a connection not getting established

If communication between the PLC and the HMI device cannot be established, the causes could be the following:

- Integrated connection not configured
- Hardware fault: Cable not plugged in, PLC switched off, network component interrupted
- Connection is not online
- Erroneous network configuration, for example. wrong access point, Invalid IP address of a device
- Failed authentication on the PLC side (wrong or invalid password)
- Wrong or invalid connection certificate
- A connection that has already been established is broken as a result of a system function call.
- Connection resources exhausted

Connection not online

The connection can be set to not 'Online' In the Engineering System. In this case, connection establishment does not take place.

Breaking and switching the connection with a system function

A connection can be broken, established and switched by calling a system function in a script.

- "SetConnectionMode": The specified connection is established or disconnected.
- "ChangeConnection": Changes the connection parameters of an HMI connection.
  Is used to switch between PLCs at runtime.
  When "SwitchConnection" (ChangeConnection) is used, the current connection to the PLC is disconnected and an attempt is made to establish a connection to another PLC. Connection establishment to the other PLC can be rejected or the connection may later be lost.

Procedure if there is no connection

The connection is established at the start of Runtime.

If a connection cannot be established, Runtime issues a system alarm.

Requirement

Configure a screen with a screen object "Alarm view" that is shown on the HMI device in Runtime.

The system alarm for a failed connection establishment is displayed here.

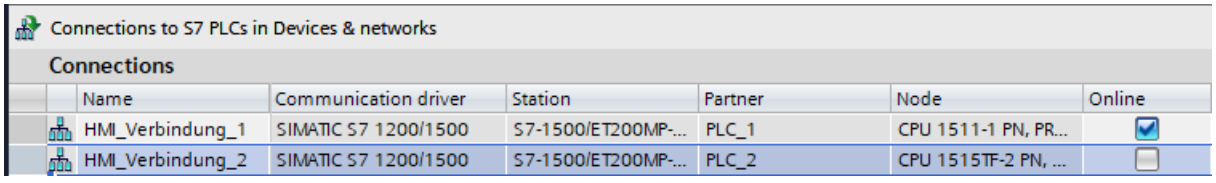
Note

If a connection is configured as not "online" or is not established by calling a system function, a system alarm is not issued. This is not an error; it is configured behavior.

The system alarm provides information on the cause of the faulty connection establishment.

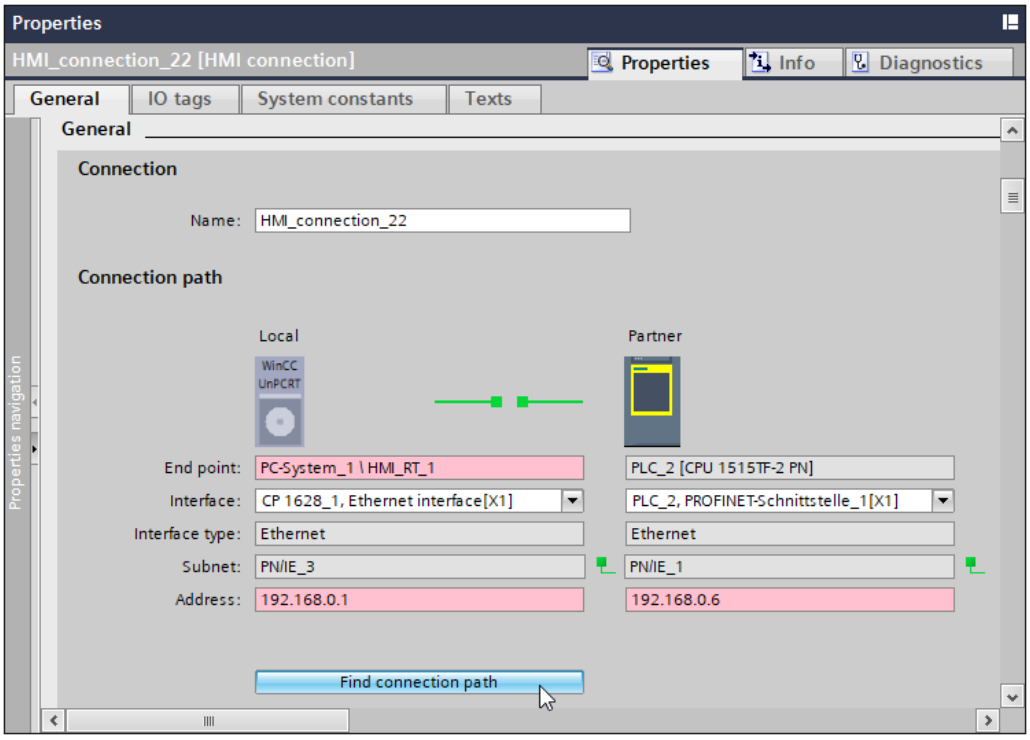
Finding the cause of the error

1. Check the cable and the plug-in connections.
2. Check whether the connection has been configured as "online".



3. Check whether a script has called the system functions "SetConnectionMode" (SetConnectionMode) or "SwitchConnection" (ChangeConnection).
   More information: Setting up switch on/switch off of a connection in runtime (Page 6993)

4. Check the network configuration.
   IP addresses within a subnet must be unique.



   If required, select the function "Find connection path" to assign the optimum connection path to the connection.

5. Check the availability and validity period of the certificate in the TIA Portal.
   The alarm text contains the name of a lapsed certificate.
   More information: Introduction to the WinCC Unified Certificate Manager (Page 7517)
   Connection establishment is not possible with PLCs that do not provide a certificate.

6. The password given for the PLC in "Device configuration > Properties > General > Access protection" must match the password that is specified for the connection.

Error messages SIMATIC S7-1200/1500

The following alarms are output if the connection cannot be established:

ID	Cause
536870948	The connection to the PLC could not be established.
537526273	The connection to an S7-1200/1500 PLC could not be established.
537526274	The S7-1200/1500 PLC is not in RUN mode.
537526275	The runtime settings of the WinCC Unified Device for controller alarms are configured for "Automatic Update", but the PLC S7-1200/1500 PLC does not support full text alarms.
537526276	The S7-1200/1500 PLC communication resources for HMI tags are overloaded.

You can find the configured alarm texts under [HMI device] > "HMI alarms > System events".

More information: Reference to system events (Page 798).

The following alarms can be displayed for certificates.

ID	Cause
536870941	The certificate was not found in the certificate memory.
536870942	Certificate expires soon.
536870943	Certificate has expired.

The alarms specify the relevant device, the name of the certificate and if applicable, the date.

14.2.2 Communicating with SIMATIC S7-300/400

14.2.2.1 Communication with SIMATIC S7-300/400

Introduction

The combined designation for the PLCs S7-300 and S7-400 is SIMATIC S7-300/400.

You can configure the following communication channel for communication between an HMI device and the SIMATIC S7-300/400 PLC:

- PROFINET

See also

Permissible data types for SIMATIC S7-300/400 (Page 7003)

Cyclic operation (Page 7007)

14.2.2.2 Permissible data types for SIMATIC S7-300/400

Permitted data types for connections with SIMATIC S7-300/400

The table lists the data types that can be used when configuring tags.

Data type	Length
BOOL	1 bit
BYTE	1 byte
CHAR	1 byte
DATE	2 bytes
DATE_AND_TIME	8 bytes
DINT	4 bytes
DWORD	4 bytes
INT	2 bytes
REAL	4 bytes

Data type	Length
S5TIME	2 bytes
STRING	(2+n) bytes, n = 0 to 254 • With SIMATIC S7-300: – Read access: 220 characters – Write access: 210 characters • ASCII • Characters from the Windows 1252 code page
TIME	4 bytes
TIME_OF_DAY, TOD	4 bytes
WORD	2 bytes

See also

Communication with SIMATIC S7-300/400 (Page 7003)
Cyclic operation (Page 7007)

14.2.2.3 Interface and communication parameters

S7-300/400 | Integrated HMI connection

PROFINET interface parameters

The following table shows the interface parameters of an integrated HMI connection:

Table 14-5 PROFINET parameters of the HMI device

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
MAC address	Specifies the MAC address for the connection type "ISO connection". This option is only available when the "Use ISO protocol" option is selected.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.

Table 14-6 PROFINET parameters of the PLC

Parameters	Description
Subnet	Specifies the subnet of the HMI connection via which the HMI device is connected to the network.
IP address	Specifies the IP address of the communication partner. This property is only available when the "Set IP address in the project" option is selected.
Subnet mask	Specifies the subnet mask. This property is only available when the "Set IP address in the project" option is selected. The subnet mask determines which part of the IP address addresses the network and which part of the IP address addresses the device.
Router address	Specifies the router address. This property is only available when the "Use router" option is selected.
PROFINET device name	Shows the PROFINET device name or specifies it. This property is only available when the "Generate PROFINET device name automatically" option is deactivated.
Converted name	Shows the name that is automatically generated from the PROFINET device name and satisfies the DNS conventions.
Device number	Shows the device number by which an IO device can be identified.

S7-300/400 | Non-integrated HMI connection

PROFINET interface parameters

The following table shows the interface parameters of a non-integrated HMI connection:

Table 14-7 PROFINET parameters of the HMI device

Parameters	Description
Interface	Specifies the communication channel.
Address	Specifies the IP address of the device.
Access point	Specifies the device name through which the communication partner can be reached.

Table 14-8 PROFINET parameters of the PLC

Parameters	Description
Address	Specifies the IP address of the device.
Expansion slot	Specifies the number of the expansion slot of the PLC to be addressed.
Rack	Specifies the rack number of the PLC to be addressed.
Cyclic operation	Enables/disables cyclic operation.

14.2.2.4 Configuring a connection via "Named connections"

Introduction

For setting up a logical connection, one of the symbolic connection names listed in the "Connection name" field is assigned to a selected application name.

The symbolic connection names and application names are configured in STEP 7.

NAMED CONNECTIONS can only be configured for integrated connections.

Note

With routed HMI connections between WinCC RT Unified and the S7-300/400 PLCs, the use of "Named Connection" is mandatory, independent of the router.

Requirements

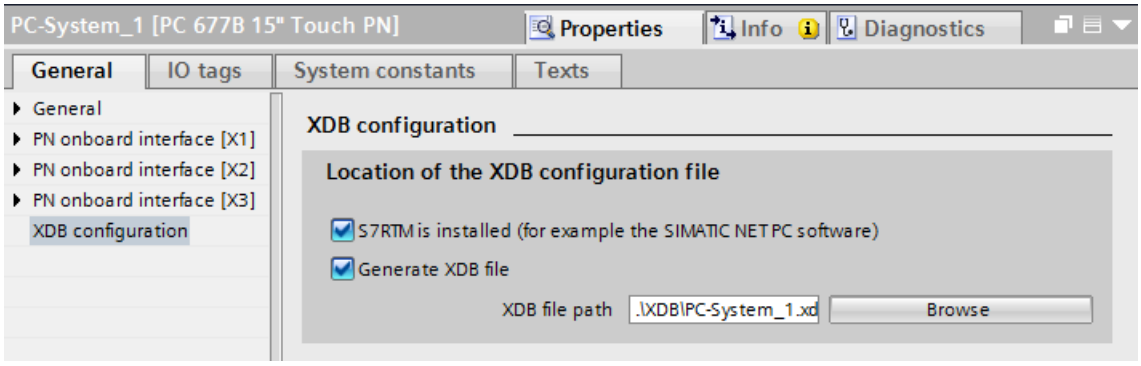
A "Named connection" has been created in STEP 7.

The following communication partners are configured in the "Devices & Networks" editor:

- SIMATIC PC with WinCC RT Professional
- SIMATIC S7-300/400

Procedure

1. Double-click the "Devices & Networks" item in the project tree.
   The available communication partners in the project are displayed in the network view.
2. Click on the HMI device in the "Network view".
3. Select the entry "S7RTM is installed" in the Inspector window under "Parameters > XDB configuration".



   If you have not installed S7RTM, select "Generate XDB file". Then select the path in which the XDB file will be stored.
4. Click the "Connections" button and select "HMI connection" for the connection type.
5. Click the PROFINET interface of the PLC and drag and drop a connection to the PROFINET interface of the PC.

6. Click the connecting line.
7. Click "Highlight HMI connection" and select the HMI connection.
   The connection is displayed graphically in the Inspector window.
8. Select the "NAMED CONNECTION" interface in the Inspector window under "Properties > General > Connection path > Interface" of WinCC RT Professional.

Note

The application and connection name can also be entered manually, for e.g. if an XDB file does not exist for the symbolic connection name or if the project is to be transferred to another computer. It is necessary to check the correct writing of the name in STEP 7 because there is no name validation in CS mode.

14.2.2.5 Cyclic operation

Basics of cyclic operation

Operating principle of cyclic operation

When the "Cyclic operation" option is enabled, the HMI device sends a telegram to the PLC at the beginning of communication indicating that certain tags are required on a recurring basis.

The PLC then always transmits the data at the same cyclic interval. This saves the HMI device from having to output new requests for the data.

When cyclic operation is disabled, the HMI device sends a request whenever information is required.

Advantages and properties of cyclic operation

The list below shows the advantages and properties of "Cyclic operation" option:

- Cyclic operation reduces the data transmission load at the HMI device. The PLC resources are used to reduce the load on the HMI device.
- The PLC only supports a certain number of cyclic services. The HMI device handles the operation if the PLC cannot provide any further resources for cyclic services.
- The HMI device generates the cycle if the PLC does not support cyclic operation.
- Screen tags are not integrated into cyclic operation.
- Cyclic operation is only set up at the restart of Runtime.
- The HMI device transfers several jobs to the PLC if cyclic operation is enabled, depending on the PLC.
- The HMI device only transfers one job at a time to the PLC if cyclic operation is disabled.

See also

Configuring cyclic operation (Page 7008)

Configuring cyclic operation

Introduction

You configure cyclic operation for an HMI connection at an HMI connection in the "Connections" editor of the HMI device.

Requirement

- The devices and networks are configured.
- An HMI connection is created in the "Connections" editor.

Procedure

To enable an HMI connection for cyclic operation, follow these steps:

1. Double-click "Connections" in the project tree below your HMI device.
   The "Connections" editor opens.
2. Select the desired HMI connection.
   The parameters of the connection are displayed in the graphic overview.
3. Activate "PLC > Cyclic operation".

Result

The PLC then always transmits the required data at the same cyclic interval.

See also

Basics of cyclic operation (Page 7007)

14.2.2.6 Troubleshooting for SIMATIC S7-300/400

Causes of a faulty connection

Causes for a connection not getting established

If communication between the PLC and the HMI device cannot be established, the causes could be the following:

- Integrated connection not configured
- Hardware fault: Cable not plugged in, PLC switched off, network component interrupted

- Connection is not online
- Fault network configuration, for example, invalid IP address of a device
- Failed authentication on the PLC side (wrong or invalid password)
- A connection that has already been established is broken as a result of a system function call.
- Connection resources exhausted

Connection not online

The connection can be set to not 'Online' In the Engineering System. In this case, connection establishment does not take place.

Breaking and switching the connection with a system function

A connection can be broken, established and switched by calling a system function in a script.

- "SetConnectionMode": The specified connection is established or disconnected.
- "ChangeConnection": Changes the connection parameters of an HMI connection.
  Is used to switch between PLCs at runtime.
  When "SwitchConnection" (ChangeConnection) is used, the current connection to the PLC is disconnected and an attempt is made to establish a connection to another PLC. Connection establishment to the other PLC can be rejected or the connection may later be lost.

Procedure if there is no connection

The connection is established at the start of Runtime.

If a connection cannot be established, Runtime issues a system alarm.

Requirement

Configure a screen with a screen object "Alarm view" that is shown on the HMI device in Runtime.

The system alarm for a failed connection establishment is displayed here.

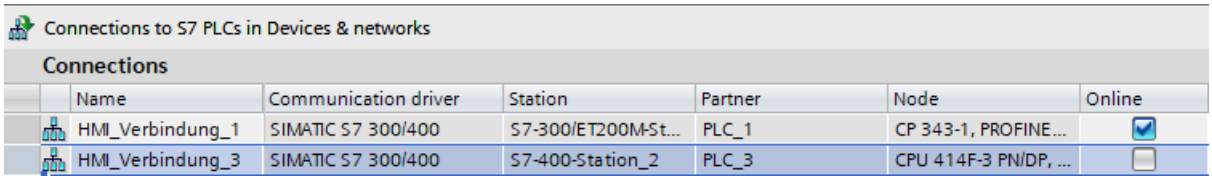
Note

If a connection is configured as not "online" or is not established by calling a system function, a system alarm is not issued. This is not an error; it is configured behavior.

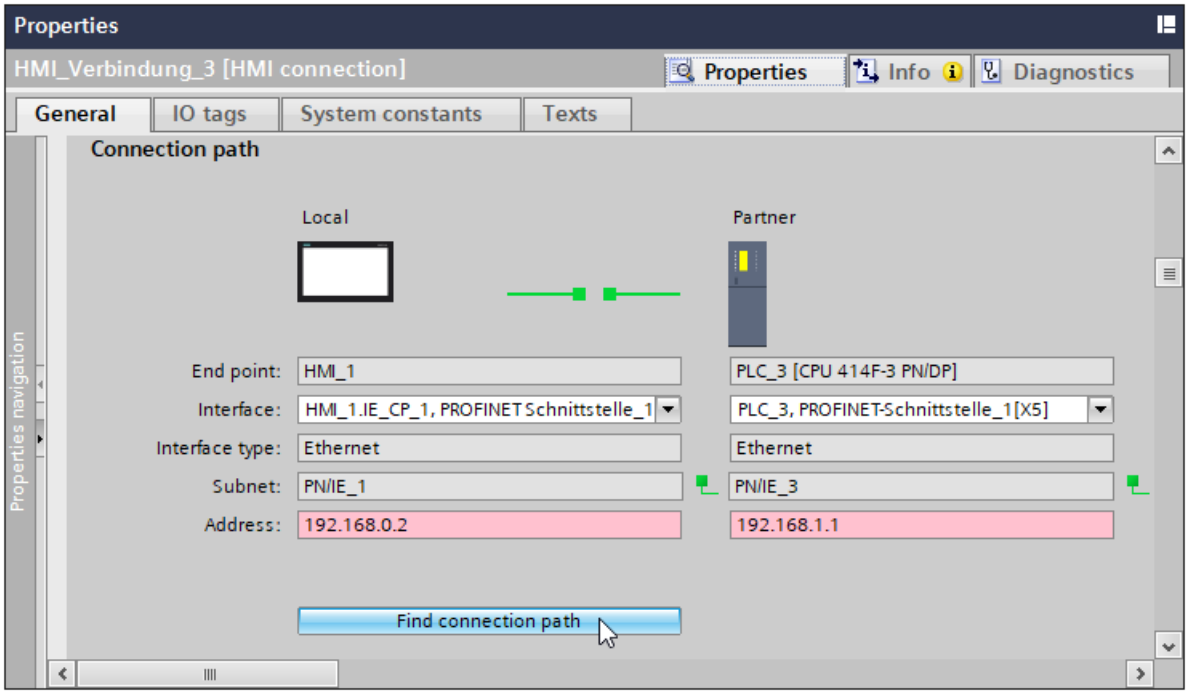
The system alarm provides information on the cause of the faulty connection establishment.

Finding the cause of the error

1. Check the cable and the plug-in connections.
2. Check whether the connection has been configured as "online".



3. Check whether a script has called the system functions "SetConnectionMode" (SetConnectionMode) or "SwitchConnection" (ChangeConnection).
   More information: Setting up switch on/switch off of a connection in runtime (Page 6993)
4. Check the network configuration.
   IP addresses within a subnet must be unique.



   If required, select the function "Find connection path" to assign the optimum connection path to the connection.
5. The password given for the PLC in "Device configuration > Properties > General > Protection" must match the password that is specified for the connection.

SIMATIC S7-300/400 error messages

The following alarms are output if the connection cannot be established:

ID	Cause
536870948	The connection to the PLC could not be established.
537526273	The connection to a S7-300/400 PLC could not be established.

ID	Cause
537526274	The S7-300/400 PLC is not in the RUN mode.
537526275	The runtime settings of the WinCC Unified Device for controller alarms are configured for "Automatic Update" but the PLC S7-300/400 PLC does not support full text alarms.
537526276	The communication resources of the S7-300/400 PLC for HMI tags are overloaded.

You can find the configured alarm texts under [HMI device] > "HMI alarms > System events".

More information: Reference to system events (Page 798).

14.3 Communication with other devices

14.3.1 Communication with WinCC Unified Open Pipe

WinCC Unified Open Pipe is an Openness concept based on pipe technology to connect a customer application to WinCC Unified RT.

The documentation is available on the Online Support: SIMATIC HMI WinCC Unified Open Pipe (https://support.industry.siemens.com/cs/ww/en/view/109803794/143390792203).

14.4 OPC UA - Open Platform Communications

14.4.1 Introduction

14.4.1.1 Principle

OPC is a standardized manufacturer-independent software interface for data exchange in automation engineering.

OPC UA is the technology succeeding OPC. OPC UA is platform-independent and supports different protocols as communication medium.

14.4.1.2 OPC UA specifications and compatibility

Overview

OPC UA specifies interfaces to gain access to the following objects in WinCC Unified:

- Process values (OPC UA 1.04)
- Tag-based alarms (OPC UA 1.04)

For detailed information about the individual OPC specifications, refer to the website of the OPC Foundation (http://www.opcfoundation.org).

Compatibility

Interoperability with OPC products from other manufacturers is guaranteed by participation in "OPC Interoperability Workshops".

14.4.2 Using OPC UA certificates

14.4.2.1 Introduction to OPC UA certificates

Introduction

Communication between an OPC UA server and its OPC UA clients that is protected by certificates requires the following:

- A valid OPC UA server certificate is installed on the server and a valid OPC UA client certificate is installed on the clients.
- The client devices trust the OPC UA server certificate and vice versa.

The type of the certificate used determines how the trust is established between OPC UA server and OPC UA clients:

- When a communication partner uses a certificate issued by a certificate authority (CA) and the other communication partners trust the root certificate of the certificate authority, they automatically trust the OPC UA certificate.

  Note

  Support of external certificate authorities

  The OPC UA certificate of a Unified device cannot be issued by an external certificate authority. The WinCC Unified Certificate Manager tool is required to create the root certificate and the OPC UA certificate.

- When a communication partner uses a self-signed OPC UA certificate, the other communication partners must explicitly trust this certificate.

  Note

  Restriction for self-signed Unified OPC UA server and client certificates

  A self-signed default certificate is generated for the Unified OPC UA server:

  - Unified PC: When installing Runtime on the PC
  - Unified Comfort Panel: When starting Runtime if no OPC UA server certificate is found in the certificate store

  In order not to use a certificate issued by a certification authority for a Unified OPC UA server, use this certificate.

  The use of a self-signed OPC UA client certificate is not possible for Unified OPC UA clients.

How you proceed to create the trust relationship on a Unified device also depends on whether you are using the Unified device as an OPC UA server or client.

If you are using a Unified device as a client, the engineering system also acts as an OPC UA client during configuration of the device.

Provision of the certificates

The following sections describe how to provide the certificates:

	Used as	Section
Unified PC	OPC UA server	Using root certificates (Unified PC as OPC UA server) (Page 7014) Using self-signed certificates (Unified PC as OPC UA server) (Page 7016)
	OPC UA client	Using root certificates (Unified PC as OPC UA client) (Page 7017)
Unified Comfort Panel	OPC UA server	Using root certificates (UCP as OPC UA server) (Page 7019) Using self-signed certificates (UCP as OPC UA server) (Page 7021)
	OPC UA client	Using root certificates (UCP as OPC UA client) (Page 7022) Using self-signed certificates (UCP as OPC UA client) (Page 7024)
Engineering System	OPC UA client	Providing certificates for the engineering systems as OPC UA client (Page 7025)

Unified tools

When you use certificates issued by a certificate authority, the following tools support you in providing the certificates:

Task	Device	Tool
Creating the root certificate of the Unified OPC UA device (if not already done)	Unified PC that is used as certification authority for the Unified OPC UA device	WinCC Unified Certificate Manager
Creating a Unified OPC UA certificate		
Installing the • Unified root certificate (if not yet done) • Unified OPC UA certificate	Unified PC that is used as OPC UA device	WinCC Unified Certificate Manager
	Unified Comfort Panel that is used as OPC UA device	Control Panel > Security > Certificates
Distribute Unified root certificate and its CRL file to the OPC UA communication partners	Unified PC that is used as certificate authority for the Unified OPC UA device[1]	WinCC Unified Certificate Manager
Import the root certificate of the OPC UA communication partner and its CRL file	Unified PC that is used as OPC UA device	SIMATIC Runtime Manager
	Unified Comfort Panel that is used as OPC UA device	Control Panel > Security > Certificates

[1] If the OPC UA device is a Unified PC, you can alternatively distribute the root certificate and CRL file directly on the device using SIMATIC Runtime Manager.

When you use self-signed OPC UA certificates, the following tools support you in providing the certificates:

Task	Device	Tool
Importing or trusting the certificate of the communication partner	OPC UA Unified PC	SIMATIC Runtime Manager
	OPC UA Unified Comfort Panel	Control Panel > Security > Certificates

Note

Operation of the Certificate Manager and Runtime Manager

For more detailed information on operating the Certificate Manager and the Runtime Manager, refer to the Runtime online help.

Certificate store of Unified devices

The OPC UA certificates are stored in the certificate store of the Unified device.

For Unified PC: `C:\ProgrammData\SCADAProjects\certstore`

14.4.2.2 Providing certificates on a Unified PC

Using a Unified PC as OPC UA server

Using root certificates (Unified PC as OPC UA server)

This section describes how you provide the certificates for the following case:

- A Unified PC is used as OPC UA server.

Note

Operating Certificate Manager and Runtime Manager

For more detailed information on operating the Certificate Manager and the Runtime Manager, refer to the Runtime online help.

Requirement

- A root certificate was generated on the Unified PC that serves as certificate authority.

Sequence

The following steps are included in providing the certificates:

1. Generate the OPC UA server certificate and export its certificate configuration.
2. Install the certificate configuration on the Unified OPC UA server.
3. Trust the OPC UA client on the Unified OPC UA server.
4. Trust the Unified OPC UA server on the OPC UA client.

Generating the server certificate and exporting the certificate configuration

1. Open the Certificate Manager on the Unified PC that serves as certificate authority.
2. Generate a OPC UA server certificate for the Unified PC that is used as OPC UA server.
3. Export the certificate configuration to an external data storage medium.
   This step can be omitted when the device that serves as certificate authority is also used as OPC UA server.

Installing the certificate configuration on the Unified OPC UA server

1. Connect the Unified PC that is used as OPC UA server to the external data storage medium.
   This step can be omitted when the device that serves as certificate authority is also used as OPC UA server.
2. Install the certificate configuration on the PC using the Certificate Manager.

The following certificates are installed:

- The root certificate including CRL file
- The OPC UA server certificate

Trusting the OPC UA client on the Unified OPC UA server

1. Save the root certificate of the OPC UA client and its CRL file (Certificate Revocation List) to an external data storage medium.
2. Connect the Unified PC that is used as OPC UA server to the external data storage medium.
3. Open the Runtime Manager on the Unified PC.
4. Import the root certificate of the OPC UA client.
   The root certificate is imported and classified as trusted.
5. Import the associated CRL file.

The Unified OPC UA server trusts the OPC UA client certificate when the next connection attempt is made.

Trusting the Unified OPC UA server on the OPC UA client

1. Ppen the Certificate Manager on the Unified PC that serves as the certification authority of the Unified OPC UA server.
2. In the Certificate Manager, export the root certificate and its CRL file (Certificate Revocation List) to an external data storage medium.

Note

Alternative

On the Unified PC that is used as OPC UA server, use the Runtime Manager to export the root certificate and CRL file.

3. Connect the OPC UA client to the external data storage medium.
4. Copy both files in the certificate store of the OPC UA client into the folder for trusted certificates. To do this, proceed as described in the application help of the client.

The OPC UA client accepts the Unified OPC UA server certificate when the next connection attempt is made.

Using self-signed certificates (Unified PC as OPC UA server)

This section describes how you provide the certificates for the following case:

- A Unified PC is used as OPC UA server.
- The OPC UA certificates of the OPC UA server and the client are self-signed.

Note

Operating the Runtime Manager

For more detailed information on operating the Runtime Manager, refer to the Runtime online help.

Sequence

1. Trust the OPC UA client on the Unified PC.
2. Trust the self-signed default certificate of the Unified OPC UA server on the OPC UA client.

Trusting the OPC UA client on the Unified OPC UA server

After the first connection attempt

If a connection attempt has already been made between the client and server, the self-signed OPC UA client certificate is available on the Unified PC in the certificate store in the "untrusted" folder.

Follow these steps:

1. Open the Runtime Manager on the Unified PC.
2. Trust the OPC UA client certificate in the Runtime Manager.

The certificate is moved to the "trusted" folder in the certificate store of the Unified PC. The Unified PC accepts the OPC UA client certificate when the next connection attempt is made.

Before the first connection attempt

To trust the self-signed certificate before a connection has been established between server and client, follow these steps:

1. Save the certificate of the OPC UA client to an external data storage medium.
2. Connect the Unified PC to the external data storage medium.
3. Open the Runtime Manager on the Unified PC.
4. Import the OPC UA client certificate.

During the import, the certificate is automatically copied to the "trusted" folder of the certificate store. The Unified PC trusts the OPC UA client certificate when the next connection attempt is made.

Trusting the Unified OPC UA server on the OPC UA client

You use the self-signed default certificate of the Unified OPC UA server.

After the first connection attempt

If a connection attempt has already been made between the client and server, the self-signed OPC UA server certificate is available on the client in the certificate store in the rejected certificates.

Copy the certificate to the certificate store for trusted certificates. To do this, proceed as described in the application help of the client.

The OPC UA client accepts the Unified OPC UA server certificate when the next connection attempt is made.

Before the first connection attempt

To trust the self-signed certificate before a connection has been established between server and client, follow these steps:

1. Copy the self-signed OPC UA server certificate on the Unified PC from the following folder to an external data storage medium:
   `<Certificate store>own\certs`
2. Connect the OPC UA client to the external data storage medium.
3. Copy the certificate to the certificate store for trusted certificates. To do this, proceed as described in the application help of the client.

The OPC UA client accepts the OPC UA server certificate when the next connection attempt is made.

Using a Unified PC as OPC UA client

Using root certificates (Unified PC as OPC UA client)

This section describes how you provide the certificates for the following case:

- A Unified PC is used as OPC UA client.
- The OPC UA certificates of the OPC UA server and the client are issued by a certificate authority.

Note

Operating Certificate Manager and Runtime Manager

For more detailed information on operating the Certificate Manager and the Runtime Manager, refer to the Runtime online help.

Requirement

- A root certificate was generated on the Unified PC that serves as certificate authority.

Sequence

The following steps are included in providing the certificates:

1. Generate the OPC UA client certificate and export its certificate configuration.
2. Install the certificate configuration on the OPC UA client.
3. Trust the OPC UA server on the Unified OPC UA client.
4. Trust the Unified OPC UA client on the OPC UA server.

Generating the client certificate and exporting the certificate configuration

1. Open the Certificate Manager on the Unified PC that serves as certificate authority.
2. Generate a OPC UA client certificate for the Unified PC that is used as OPC UA client.
3. Export the certificate configuration to an external data storage medium.
   This step can be omitted when the device that serves as certificate authority is also used as OPC UA client.

Installing the certificate configuration on the Unified OPC UA client

1. Connect the Unified PC that is used as OPC UA client to the external data storage medium.
   This step can be omitted when the device that serves as certificate authority is also used as OPC UA client.
2. Install the certificate configuration on the PC using the Certificate Manager.

The following certificates are installed:

- The root certificate including CRL file
- The OPC UA client certificate.

Trusting the OPC UA server on the Unified OPC UA client

1. Save the root certificate of the OPC UA server and its CRL file (Certificate Revocation List) to an external data storage medium.
2. Connect the Unified PC that is used as OPC UA client to the external data storage medium.
3. Open the Runtime Manager on the Unified PC.
4. Import the root certificate of the OPC UA server.
5. Import the associated CRL file.

The Unified OPC UA client trusts the OPC UA server certificate when the next connection attempt is made.

Trusting the Unified OPC UA client on the OPC UA server

1. Open the Certificate Manager on the Unified PC that serves as the certificate authority of the Unified OPC UA client.
2. In the Certificate Manager, export the root certificate and its CRL file (Certificate Revocation List) to an external data storage medium.

   Note

   Alternative

   On the Unified PC that is used as OPC UA client, use the Runtime Manager to export the root certificate and CRL file.

3. Connect the OPC UA server to the external data storage medium.
4. Copy both files to the certificate store for trusted certificates. To do this, proceed as described in the application help of the server.

The OPC UA server accepts the Unified OPC UA client certificate when the next connection attempt is made.

14.4.2.3 Providing certificates on a Unified Comfort Panel

Use the Unified Comfort Panel as an OPC UA server

Using root certificates (UCP as OPC UA server)

This section describes how you provide the certificates for the following case:

- A Unified Comfort Panel is used as OPC UA server.
- The OPC UA certificates of the OPC UA server and the client are issued by a certificate authority.

Note

Operating the Certificate Manager

For more detailed information on operating the Certificate Manager, refer to the Runtime online help.

Requirement

- A root certificate was generated on the Unified PC that serves as certificate authority.

Sequence

The following steps are included in providing the certificates:

1. Generate the OPC UA server certificate and export its certificate configuration.
2. Install the certificate configuration on the Unified OPC UA server.
3. Trust the OPC UA client on the Unified OPC UA server.
4. Trust the Unified OPC UA server on the OPC UA client.

Generating the server certificate and exporting the certificate configuration

1. Open the Certificate Manager on the Unified PC that serves as certificate authority.
2. Generate an OPC UA server certificate for the Unified Comfort Panel that is used as OPC UA server.
3. Export the certificate configuration to an external data storage medium.
   The root certificate, its CRL file and the OPC UA Server certificate are exported in encrypted format.

Installing the certificate configuration on the Unified OPC UA server

1. If Runtime starts on the Panel that is used as OPC UA server and no OPC UA server certificate is found, a self-signed default certificate is generated.
   Delete the default certificate. Follow these steps:
   - Select "Security > Certificates" in the Control Panel on the Panel.
   - Select the "My Certificates" entry from the "Certificate store" list.
   - Select the OPC UA server default certificate.
   - Click "Delete".
2. Connect the Panel to the external data storage medium onto which you have exported the certificate configuration.
3. Install the root certificate and the OPC UA server certificate.
   For both certificates, follow these steps:
   - Select "Security > Certificates" in the Control Panel on the Panel.
   - Click the "Import" button.
   - In the "Import certifcates" dialog, select the certificate from the external storage medium.
   - Enter the password and the iteration specified during the export in the Certificate Manager.
   - Confirm your entries.

The following certificates are installed:

- The root certificate including CRL file
- The OPC UA server certificate

Trusting the OPC UA client on the Unified OPC UA server

1. Save the root certificate of the OPC UA client and its CRL file (Certificate Revocation List) to an external data storage medium.
2. Connect the Unified Comfort Panel to the external data storage medium.
3. Select "Security > Certificates" in the Control Panel on the Panel.
4. Click the "Import" button.
5. In the "Import certifcates" dialog, select the certificate from the external storage medium.
6. Confirm your entries.

The root certificate and its CRL are imported and classified as trusted.

The Unified OPC UA server trusts the OPC UA client certificate when the next connection attempt is made.

Trusting the Unified OPC UA server on the OPC UA client

1. Open the Certificate Manager on the Unified PC that serves as the certificate authority of the panel.
2. In the Certificate Manager, export the root certificate and its CRL file (Certificate Revocation List) to an external data storage medium.
3. Connect the OPC UA client to the external data storage medium.
4. Copy both files to the certificate store for trusted certificates. To do this, proceed as described in the application help of the client.

The OPC UA client accepts the Unified OPC UA server certificate when the next connection attempt is made.

Using self-signed certificates (UCP as OPC UA server)

This section describes how you provide the certificates for the following case:

- A Unified Comfort Panel is used as OPC UA server.
- The OPC UA certificates of the OPC UA server and the client are self-signed.

Sequence

1. Trust the OPC UA client on the Unified Comfort Panel.
2. Trust the self-signed default certificate of the Unified OPC UA server on the OPC UA client.

Trusting the OPC UA client on the Unified OPC UA server

After the first connection has been established between server and client, the self-signed OPC UA client certificate is available on the Unified Comfort Panel. The Panel does not yet trust the certificate. Follow these steps:

1. Select "Security > Certificates" in the Control Panel on the Panel.
2. Select the "Other Certificates" entry from the "Certificate store" list.

3. Select the OPC UA client certificate.
   The certificate has the status "Untrusted".
4. Click "Trust".

The Unified Comfort Panel accepts the OPC UA client certificate when the next connection attempt is made.

Trusting the Unified OPC UA server on the OPC UA client

You use the self-signed default certificate of the Unified OPC UA server.

After the first connection is established between the server and client, the self-signed certificate of the OPC UA server is available on the OPC UA client. The client does not yet trust the certificate.

Trust the Unified OPC UA server certificate on the OPC UA client.

The client accepts the server certificate the next time it attempts to connect.

Use the Unified Comfort Panel as an OPC UA client

Using root certificates (UCP as OPC UA client)

This section describes how you provide the certificates for the following case:

- A Unified Comfort Panel is used as OPC UA client.
- The OPC UA certificates of the OPC UA server and the client are issued by a certificate authority.

Note

Operating the Certificate Manager

For more detailed information on operating the Certificate Manager, refer to the Runtime online help.

Requirement

- A root certificate was generated on the Unified PC that serves as certificate authority.

Sequence

The following steps are included in providing the certificates:

1. Generate the OPC UA client certificate and export its certificate configuration.
2. Install the certificate configuration on the Unified OPC UA client.
3. Trust the OPC UA server on the Unified OPC UA client.
4. Trust the Unified OPC UA client on the OPC UA server.

Generating the client certificate and exporting the certificate configuration

1. Open the Certificate Manager on the Unified PC that serves as certificate authority.
2. Generate an OPC UA client certificate for the Unified Comfort Panel that is used as OPC UA client.
3. Export the certificate configuration to an external data storage medium.

Installing the certificate configuration on the Unified OPC UA client

1. Connect the Panel to the external data storage medium onto which you have exported the certificate configuration.
2. Install the root certificate and the OPC UA Client certificate.
   For both certificates, follow these steps:
   - Select "Security > Certificates" in the Control Panel on the Panel.
   - Click the "Import" button.
   - In the "Import certifcates" dialog, select the certificate from the external storage medium.
   - Enter the password and the iteration specified during the export in the Certificate Manager.
   - Confirm your entries.

   The following certificates are installed:
   - The root certificate including CRL file
   - The OPC UA client certificate.

Trusting the OPC UA server on the Unified OPC UA client

1. Save the root certificate of the OPC UA server and its CRL file (Certificate Revocation List) to an external data storage medium.
2. Connect the Unified Comfort Panel that is used as Unified OPC UA client to the external data storage medium.
3. Select "Security > Certificates" in the Control Panel on the Panel.
4. Click the "Import" button.
5. In the "Import certificates" dialog, select the certificate from the external storage medium.
6. Confirm your entries.

The root certificate and its CRL are imported and classified as trusted.

The Unified OPC UA client trusts the OPC UA server certificate when the next connection attempt is made.

Trusting the Unified OPC UA client on the OPC UA server

1. Open the Certificate Manager on the Unified PC that serves as the certificate authority of the panel.
2. In the Certificate Manager, export the root certificate and its CRL file (Certificate Revocation List) to an external data storage medium.
3. Connect the OPC UA server to the external data storage medium.
4. Copy both files to the certificate store for trusted certificates. To do this, proceed as described in the application help of the server.

The OPC UA server accepts the Unified OPC UA client certificate when the next connection attempt is made.

Using self-signed certificates (UCP as OPC UA client)

This section describes how you provide the certificates for the following case:

- A Unified Comfort Panel is used as OPC UA client.
- The OPC UA server certificate is self-signed.
- The OPC UA client certificate is issued by a certificate authority.

Note

No use of self-signed Unified OPC UA client certificates

The use of a self-signed OPC UA client certificate is not possible for a Unified OPC UA client.

Sequence

1. Generate your own self-signed OPC UA client certificate for the Unified Comfort Panel.
2. Install the self-signed certificate on the Panel.
3. Trust the OPC UA server on the Unified OPC UA client.
4. Trust the Unified OPC UA client on the OPC UA server.

Trusting the OPC UA server on the Unified OPC UA client

After the first connection has been established between the server and client, the self-signed OPC UA server certificate is available on the Unified Comfort Panel. The Panel does not yet trust the certificate. Follow these steps:

1. Select "Security > Certificates" in the Control Panel on the Panel.
2. Select the "Other Certificates" entry from the "Certificate store" list.
3. Select the OPC UA server certificate.
   The certificate has the status "Untrusted".
4. Click "Trust".

The Unified Comfort Panel accepts the OPC UA server certificate when the next connection attempt is made.

Trusting the Unified OPC UA client on the OPC UA server

1. Open the Certificate Manager on the Unified PC that serves as the certificate authority of the panel.
2. In the Certificate Manager, export the root certificate and its CRL file (Certificate Revocation List) to an external data storage medium.
3. Connect the OPC UA server to the external data storage medium.
4. Copy both files to the certificate store for trusted certificates. To do this, proceed as described in the application help of the server.

The OPC UA server accepts the Unified OPC UA client certificate when the next connection attempt is made.

14.4.2.4 Providing certificates for the engineering systems as OPC UA client

Engineering system as an OPC UA client

If the engineering system acts as an OPC UA client, provide the certificates as follows:

- When the connection is first established, the client certificate is created automatically and transferred to the server.

  Note

  Trust the client certificate

  Move the client certificate on the server from the "untrusted" folder to the "trusted" folder.

- The engineering system automatically receives the server certificate and trusts it without your having to take any action.

14.4.3 WinCC Unified OPC UA server

14.4.3.1 General information about Unified OPC UA servers

Using the WinCC Unified OPC UA server

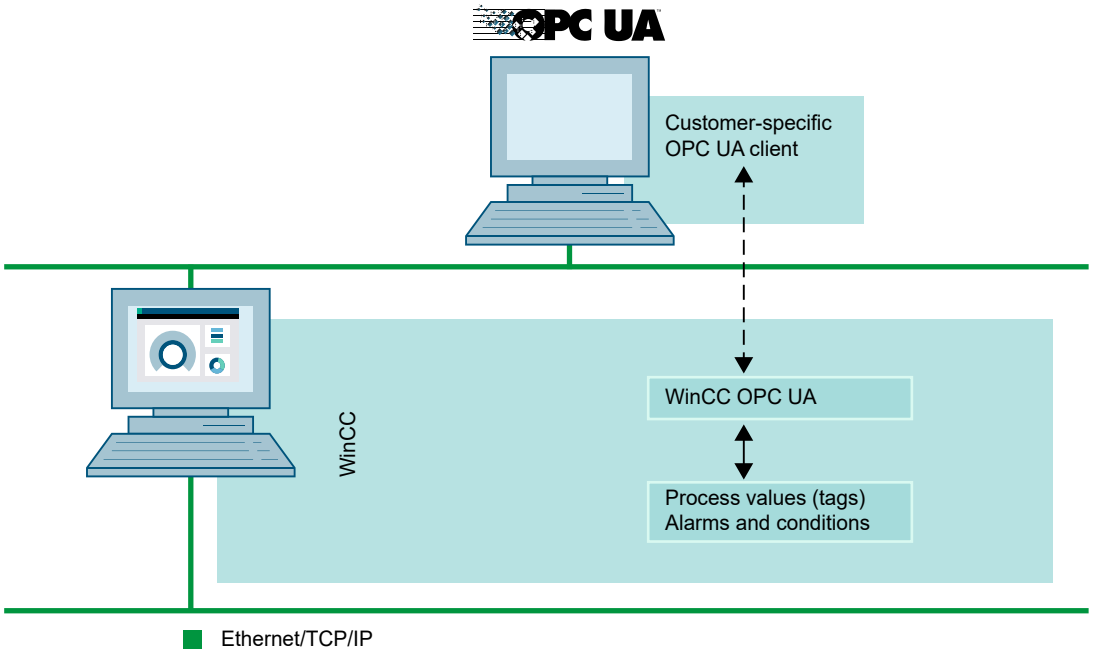
Introduction

Servers are available for the following OPC UA interfaces in WinCC Unified:

- OPC Unified Architecture: Access to the data management of WinCC Unified.

OPC UA communications concept of WinCC Unified

The figure below shows the OPC UA communication concept of WinCC Unified:



Licensing

OPC server	Licensing
WinCC OPC UA Server	A valid Runtime license for WinCC Unified

Requirements for use

Windows firewall settings (Unified PC as OPC UA server)

After installation of WinCC Unified, the Windows firewall settings of the OPC UA servers of WinCC Unified are correctly configured.

If OPC UA clients access OPC UA servers in different subnets, you must adapt the configuration of the permitted network areas to the OPC UA servers.

TIA Portal settings

In order to work with OPC UA in WinCC Unified, the OPC UA server must be enabled in the TIA Portal.

To do to this, select the "Operate as OPC UA server" option in the runtime settings under "OPC UA server > General". As soon as the option is selected, you can make additional settings.

More information is available under Configuring a Unified OPC UA server (Unified PC) (Page 7039) and Using the Unified Comfort Panel as OPC UA server (Page 7042).

Operating principle of the OPC UA server

How it works

The OPC UA server provides the following values:

- Process values
- Tag-based alarms

The OPC UA server supports only the "UA-TCP UA-SC UA Binary" communication profile. The used port number is adjustable.

You can find more information about configuration of the OPC UA server here:

- For Unified PC: In the section "Configuring a Unified OPC UA server (Unified PC) (Page 7039)"
- For Unified Comfort Panel: In the section "Configuring the Unified OPC UA server (UCP) (Page 7042)"

Supported specifications

OPC UA Architecture is a specification for the transmission of process values and alarms. The OPC UA server supports the OPC UA specification 1.04.

For more information about supported OPC UA functions, refer to "OPC UA specifications and compatibility (Page 7011)".

Starting the OPC UA server

The OPC UA server is run automatically when Runtime is started after successful configuration in the TIA Portal.

URL of the OPC UA server

You can reach the OPC UA server via the following URL:

- "opc.tcp://[HostName]:[Port]"

Parameter	Description
HostName	Placeholder for the computer name. Is used automatically.
Port	Specifies the port number. "4890" is set by default. Do not use a port number that is already assigned to another application.

Discovery Server (Unified PC as OPC UA server)

The "Discovery Server" is available by the OPC foundation. The "Discovery Server" is by default installed on the HMI device as Windows service.

The "Discovery Server" makes information available to UA clients about OPC UA servers that are subscribed to the "Discovery Server".

The OPC UA server registers itself at the start of Runtime to no, one or more configured and available "Discovery Servers" depending on its configuration. Registration is then repeated cyclically. When you end Runtime, the OPC UA server is automatically logged off from the "Discovery Server".

You can find information on disabling the OPC UA Local Discovery Server in Siemens Industry Online Support (https://support.industry.siemens.com/cs/document/109749461/how-do-you-disable-the-opcua-local-discovery-server-service-for-wincc-v7-and-wincc-tia-portal-?dti=0&lc=en-WW) (Entry ID 109749461).

Supported languages in the WinCC Unified address area

The OPC UA server supports the WinCC Unified address area in the following languages:

- English

Security concept of OPC UA

Introduction

The OPC UA security concept is based on:

- Authentication and authorization of the participating applications and users
- Ensuring the integrity and confidentiality of messages exchanged between the applications.

Certificates

Certificates represent the authentication mechanism of OPC UA applications. Each application has its own instance certificate and thereby identifies itself within the public key infrastructure.

Instance certificate of the OPC UA server

Each OPC UA server for secure operation requires a separate instance certificate with a private key. The certificate is only valid on the respective computer and may be used only by the OPC UA server installed there.

When you install the server, a self-signed instance certificate of the server is created and stored in the certificate folder of the server.

The private key for this certificate is only stored in the certificate store. Access to the folder with the private key must be restricted to:

- The server itself (account of the local system)
- The system administrator

NOTICE
Restricted access to the private key folder Except for the server and the system administrator, no other users and applications may have access to the private key of the OPC UA server for security reasons. Restricted access to the private key is therefore pre-configured after installing WinCC Unified.

The instance certificate generated during installation and the associated private key can be replaced by the system administrator. In accordance with the respective security concept of the plant, the new instance certificate may be self-signed or created by a certification authority.

The certificate and the private key are stored under this folder:
"C:\ProgramData\SCADAProjects\Certstore\own".

The private key is stored in the subfolder "private".

Validation of the server instance certificate

The instance certificate of the server is validated during the start of the OPC UA server. If the public key or the private key cannot be found or if the certificate is invalid (for example, because it has expired or is corrupt), the server stops and an appropriate entry is made in the trace log.

Trusted client certificates

The OPC UA server supports secure communication to trusted clients only. A client is trusted:

- when the client has a valid self-signed certificate that is located in the certificate store of trusted certificates of the OPC UA server
- or if the valid client certificate was issued by a certification authority. The valid certificate of the certification authority must be located in the certificate store of the trustworthy certification authorities of the OPC UA server. In this case, only the certificate from the certification authority is required. The instance certificate of the client does not need to be in the certificate store of trusted certificates.

Note

Certificates from the memory of the certification authorities are not automatically trusted.

For a certification authority to be trusted, its certificate must be located in the memory for trusted certificates.

Trusted certificates are stored in the directory
"C:\ProgramData\SCADAProjects\Certstore\Trusted\certs".

The certificates from certificate authorities that are required for the verification of a client certificate chain are also stored in the certificate store of the certificate authorities.

Client certificates not accepted

When a OPC UA client accesses the OPC UA server without its trusted certificate, the OPC UA server rejects secure communication and copies the client certificate to the folder for rejected certificates. These certificates are stored in the directory "C:\ProgramData\SCADAProjects\Certstore\Trusted\untrusted".

To enable secure communication with this client, you will have to move the rejected certificate to the certificate memory for trusted certificates.

Supported OPC UA services and profiles

OPC UA services

The following table sets out the functionality supported by OPC UA server 1.04:

OPC UA Service Sets	Services	Comment
Discovery Service Set	FindServers GetEndpoints	-
Secure Channel Service Session Service Set	All	-
View Service Set	Browse BrowseNext	Determination of the WinCC Unified data shown: Process values and archived data
Attribute Service Set	Read Write	only WinCC Unified tags only WinCC Unified tags
Subscription Service Set	CreateSubscription SetPublishingMode Publish RePublish DeleteSubscription	
MonitoredItem Service Set	CreateMonitoredItems SetMonitoringMode DeleteMonitoredItems	only "Value" attribute of the WinCC Unified tags .EventNotifier during access to WinCC Unified alarms

OPC UA profiles and Conformance Units

The OPC UA server supports the following OPC UA 1.03 profiles without restrictions:

- 6.5.5 Base Server Behavior Facet
- 6.5.16 Standard Event Subscription Server Facet
- 6.5.131 UA TCP UA SC UA Binary
- 6.5.148 SecurityPolicy - Basic128Rsa15
- 6.5.149 SecurityPolicy - Basic256
- 6.5.150 SecurityPolicy - Basic256SHA256

The OPC UA server supports the following OPC UA profiles shown in the following table with restrictions:

Profile	"Group"	Not supported "Conformance Unit"
6.5.11 Standard DataChange Subscription Server Facet Subscription Server Facet	Monitored Item Services	ModifyMonitoredItems DeadBand Filter Monitor MinQueueSize_02
6.5.12 Enhanced DataChange Subscription Server Facet	Monitored Item Services	Monitor MinQueueSize_05
6.5.2 Core Server Facet	Attribute Services	Attribute Write Index
6.5.14 Data Access Server Facet	Data Access	Data Access Analog Data Access Multistate Data Access PercentDeadBand Data Access Semantic Changes Data Access Two State
6.5.55 Standard UA Server Profile	Attribute Services	Attribute Write StatusCode & TimeStamp
6.5.55 Standard UA Server Profile	Attribute Services	Attribute Write StatusCode & Timestamp
6.5.16 Standard Event Subscription Server Facet	Event Access	Base Info EventQueueOverflowEventType Base Info Progress Events Base Info SemanticChange Base Info System Status Base Info System Status underlying system Base Info Device Failure
6.5.17 Address Space Notifier Server Facet	Event Access	
6.5.18 A & C Base Condition Server Facet	Alarms and Conditions	
6.5.20 A & C Address Space Instance Server Facet	Alarms and Conditions	
6.5.21 A & C Enable Server Facet	Alarms and Conditions	
6.5.22 A & C Alarm Server Facet	Alarms and Conditions	A & C Comment A & C Discrete A & C OffNormal A & C SystemOffNormal A & C Trip
6.5.23 A & C Acknowledgeable Alarm Server Facet	Alarms and Conditions	

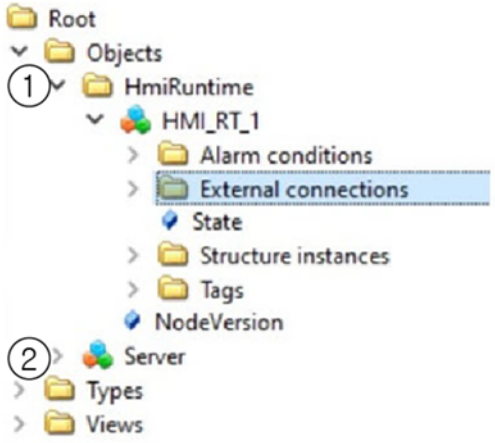
Address space of the OPC UA server

Introduction

A WinCC Unified device that is used as an OPC UA server makes the following runtime data of its system available to its OPC UA clients in its address space:

- Process values (WinCC Unified tags)
- Alarms (tag-based WinCC Unified alarms)

The address space of the OPC UA server is added below "Root > Objects" and has the following hierarchical structure:



① The folder for the Runtime system node
Folders for the alarm conditions and tags of the system are located under the node.
The structure in the tags folder corresponds to the structure of the tags in WinCC Unified.

② The server node

Mapping of the WinCC Unified tag

Connections are displayed by OPC UA objects of the "HMIConnectionType" type.

Internal and external WinCC Unified tags are displayed by OPC UA tags of the "HMISimpleTagType" type.

The following table shows the most important attributes of the OPC UA tags that represent a WinCC Unified tag. You can find the complete list of attributes in the "OPC UA Part 3 - Address Space Model 1.03 Specification" under paragraph "5.6":

Attribute	Description	Comment
NodeId	Unique designation of the WinCC Unified tag	-
BrowseName	Name of the WinCC Unified tag	-
DisplayName	Name of the WinCC Unified tag	-
Value	Tag value and status	-

Attribute	Description	Comment
DataType	OPC UA data type that corresponds to the WinCC Unified tag type, for example: • Int32; signed 32 bit value • UInt32; unsigned 32 bit value	-
AccessLevel	"CurrentRead" / "CurrentWrite"	• Correspondingly to the WinCC Unified tag configuration. • System tags "CurrentRead" only.
ValueRank	Always "Scalar"	-

Mapping of the WinCC Unified alarms

Depending on their state machine, the alarms are mapped to the following OPC UA types:

State machine of the alarm	OPC UA type
RaiseClear RaiseClearRequiresReset	HmiConditionType
RaiseOptionalClearOrAcknowledgment RaiseClearOptionalAcknowledgement RaiseClearOptionalAcknowledgementAndReset RaiseRequiresAcknowledgement RaiseClearRequiresAcknowledgement RaiseClearRequiresAcknowledgementAndReset	HmiAlarmType

Priority

For the configuration of the alarms in WinCC Unified, you select a priority between "0" and "255". The OPC UA specification defines a value range between "1" for the lowest severity and "1000" for the highest severity.

The value of the priority must therefore be selected to match the OPC UA severity. In a standard mapping, the priority "0" is assigned to the OPC UA severity "1", and the priority "255" to the OPC UA severity "1000". All other values are interpolated linearly between "0" and "1000".

Mapping the OPC UA properties

The alarm condition consists of OPC UA event properties and WinCC Unified alarm properties. The properties of the alarm condition may vary depending on the OPC UA event type.

The following table provides the most important properties of the OPC UA events and shows how the WinCC Unified alarm system provides the information.

Note

Optional properties

Optional properties are not disclosed in the server address space.

OPC UA property	Description/Mapping
For all event types:	
EventId	A unique identifier for event notification.
EventType	The NodeId of the HmiConditionType node or HmiAlarmType node
SourceNode	NodeId of the Runtime object
SourceName	Name of the Runtime object
Time	RAISETIME of the WinCC Unified alarm Time stamp when the alarm was triggered at the source.
ReceiveTime	When the server has received the event from the underlying system.
LocalTime	Information about the local time from which the event originated.
Message	EVENTTEXT of the WinCC Unified alarm, multilingual text for messages and alarms.
Severity	PRIORITY of the WinCC Unified alarm that is mapped to the OPC UA severity.
For the HmiConditionType and HmiAlarmType event types:	
ConditionId	NodeId of the condition instance
ConditionClassId	NodeId of the ProcessConditionClassType node
ConditionClassName	ProcessConditionClassType
ConditionName	NAME of the WinCC Unified alarm
BranchId	Not relevant
Retain	RETAIN of the WinCC Unified alarm TRUE for pending alarms
EanbledState/Id	ENABLESTATE of the WinCC Unified alarm TRUE for active alarms
Quality	VALUEQUALITY of the WinCC Unified alarm when the alarm became active
LastServerity	Not relevant
Comment	COMMENTS of the WinCC Unified alarm provided by the operator
ClientUserId	USER that is related to the WinCC Unified alarm
For the HmiAlarmType event type:	
AckedState	Mapped to STATE of the WinCC Unified alarm TRUE for acknowledged alarms
ConfirmedState/Id[1]	Mapped to STATE of the WinCC Unified alarm TRUE for confirmed alarms
ActiveState/Id	Mapped to STATE of the WinCC Unified alarm TRUE for active alarms
InputNode	NodeId of the tag assigned to the alarm
SupressedState/Id	Mapped to SUPPRESSIONSTATE of the WinCC Unified alarm TRUE for suppressed alarm

OPC UA property	Description/Mapping
SupressedOrShelved	Mapped to SUPPRESSIONSTATE of the WinCC Unified alarm TRUE for reset or suppressed alarms
MaxTimeShelved	Not supported

[1] Only for alarms with the state machine RaiseClearRequiresAcknowledgmentAndReset

The following table provides the configurable properties of the WinCC Unified alarms. The properties are mapped one-to-one to OPC UA event properties.

The table applies to all event types:

Optional	Property	Description
-	INSTANCEID	Instance index used to reference an active multi-instance alarm within the (configured) HmiAlarm.
-	ALARM	Pointer to the corresponding HmiAlarm
-	ALARMCLASS	Pointer to the alarm class May differ from the alarm class reference of the associated HmiAlarm.
-	ALARMCLASSSYMBOL	Symbol (abbreviation) of the referenced alarm class
-	TEXTCOLOR	Text color
-	BACKCOLOR	Background color
-	FLASHING	Flashing
-	SUPPRESSIONSTATE	Indicates whether the alarm is reset, suppressed or not suppressed.
✓	ALARMTEXT1 ... ALARMTEXT9	Additional multi-lingual texts (Text 1 to Text 9)
✓	ALARMPARAMETERVALUES1 ... ALARMPARAMETERVALUES16	Parameter value 1 to parameter value 16
-	INVALIDFLAGS	Indicator of invalid property values Bit-by-bit interpretation
✓	ORIGIN	Dynamic alarm-instance-specific name of the alarm-triggering object.
✓	AREA	Dynamic alarm-instance-specific name of the area to which the alarm-triggering object belongs.
✓	LOOPINALARM	Function that is called to navigate from the alarm control, for example, to the screen that shows the source of the alarm or to an application that provides more information.
✓	COMPUTER	Name of the machine that hosts the originator of the alarm.
✓	USERNAME	Name of the user associated with the event (operator alarms only).

Optional	Property	Description
✓	VALUE	Current value at the time when the alarm became active. Updated value at the time the alarm became inactive.
✓	VALUEQUALITY	Current quality at the time when the alarm became active.
✓	VALUELIMIT	Current limit at the time when the alarm became active. For dynamic limits: Updated limit at the time the alarm became inactive.
✓	DEADBAND	Dead zone value of the alarm condition of an analog alarm at the time when the alarm became active.
✓	CONNECTION	Reference to the corresponding HMI connection
✓	SYSTEMSEVERITY	Severity for alarm-based system voting (redundancy)
-	SOURCETYPE	Defines the alarm generation method
-	STATE	The change of the current alarm condition, including history.
-	STATETEXT	Textual representation of the alarm condition
-	CHANGEREASON	Reason for the change time, see Enumeration definition.
-	ACKTIME	Time stamp of the time when the alarm was acknowledged at the source (or the service in case the alarm source does not provide an acknowledgment).
-	CLEARTIME	Time stamp of the time when the alarm at the source became inactive (or the service, in case the alarm source does not provide date and time information).
-	RESETTIME	Time stamp of the alarm reset time (or service, in case the alarm source does not provide date and time information).
✓	LOCALTIME	Information about the local time from which the alarm originated.
-	USERRESPONSE	The type of tag that represents a property of another node.
✓	DURATION	Returns the time interval in nanoseconds between triggering of the alarm and its previous status change.

OPC UA Data Access

Tags

The WinCC Unified tags are displayed by OPC UA tags of the "HMISimpleTagType" type. Other DataAccess tag types such as "AnalogItem" or "DiscreteType" are not supported.

The OPC UA server supports read access to the OPC UA tag attributes such as "DataType" or "AccessLevel". Writing access and subscriptions are only supported for the "Value" attribute.

Inverse browsing on the OPC UA client

The functionality for inverse browsing of tags is not supported in the OPC UA server.

Alarm conditions

Communication between a WinCC Unified device which is used as an OPC UA server and its OPC UA clients includes tag-based alarms.

Availability in the address space

Based on their state machine, the configured alarms of the system running on the WinCC Unified device are mapped to OPC UA types and loaded with their properties into the address space of the OPC UA server. See section Address space of the OPC UA server (Page 7032).

OPC UA clients have read access to the alarms and their properties in the address space.

Monitoring alarms

OPC UA clients can monitor changes to the WinCC Unified alarms by subscribing to the server object or directly to the runtime system for monitoring. A client can subscribe to one object (server or runtime system) or several objects for monitoring.

When a configured alarm becomes active or a property of an active alarm changes, the OPC UA client is automatically notified.

OPC UA clients can perform the following actions for monitored alarms:

Action	Availability	Result
Acknowledge	For alarms of the OPC UA type HmiAlarmType	The alarm state is updated accordingly in the address space in the properties of the alarm. The runtime system reflects this change.
Confirm	For alarms of the OPC UA type HmiAlarmType	
Activate	For alarms of the OPC UA types HmiConditionType and HmiAlarmType	
Disable		
Shelve		
Unshelve alarm	For alarms of the OPC UA type HmiAlarmType	

Managing OPC UA server certificates

If you use a WinCC Unified device as an OPC UA server and the OPC UA communication is protected by certificates, the following applies:

- An OPC UA server certificate must be installed on the Unified device.
- The OPC UA clients must trust the OPC UA server certificate.
- The Unified device must trust the OPC UA client certificates.

Section Introduction to OPC UA certificates (Page 7012) describes how you proceed to provide the certificates required for communication.

14.4.3.2 Using the Unified PC as OPC UA server

Exporting tags

Offline export of tags using the OPC UA server

SIMATIC Runtime Manager allows you to export the tags configured for the active Runtime project to an OPC UA Nodeset using the OPC UA server. The exported data can be imported into another application, e.g. the TIA Portal, without the need for a connection to the OPC UA server.

The export makes it easier to apply an existing configuration to a new Runtime system.

Requirement

- The OPC UA server for WinCC Unified is running.
- A WinCC Unified Runtime project is running on the server.
- The following applies to the user who started the export:
  - The user has the role "SIMATIC HMI".
  - The user has the function right for read and write access to OPC UA.
- The OPC UA server certificate and the WinCC Unified OPC UA exporter certificate trust each other.
  If you have generated and installed the certificates again via the Certificate Manager, this is automatically the case.
  If you want to use the default certificates created during the Runtime installation, move the certificates so that they trust each other:

Source directory	Target directory
`C:\SCADAProjects\certstore\own\certs`	`C:\SCADAProjects\certstore\trusted\certs`

Procedure

Follow these steps to export the tags of the active Runtime using the OPC UA server:

1. Start "SIMATIC Runtime Manager".
2. Click the button  in the toolbar.
3. Configure the export settings in the "OPC UA Export" tab:
   - Select the name and the folder of the output file.
   - Type in the user name and password of the user who started the export.
4. Click "Export".

Result

You can see whether the export was successful in the "Status" field.

If the export is successful, the file is written to the specified folder.

For diagnostic purposes, a trace file is written to the following folder: [ProgramData]/Siemens/Automation/Logfiles

See also

Requirements for use (Page 7026)

Configuring a Unified OPC UA server (Unified PC)

OPC UA server

General

OPC is a standardized manufacturer-independent software interface for data exchange in automation engineering. OPC UA is the technology succeeding OPC. OPC UA is platform-independent and supports different protocols as communication medium.

To work with OPC UA in WinCC Unified, the OPC UA server must be enabled in the TIA Portal in the Runtime settings of the HMI device.

Read/write tags and register tags/alarms

When you enable the "Operate as OPC UA server" option in the HMI device, the protection for unauthorized internal and external access is downgraded.

- Enable the "Operate as OPC UA server" option.
  A security note is displayed.

After enabling the option, all other settings of the OPC UA Server will become available.

Alarms and Conditions

- To display alarm conditions in the address range of the server, select the option "Enable Alarms and Conditions on the OPC UA server".
- To disable or acknowledge alarms on the OPC UA Client, for example, select the option "Allow operation of alarms on the OPC UA Client". To enable this option, the "Enable Alarms and Conditions on the OPC UA server" option must be enabled.

Options

General

Define the following settings:

- Port
  Default value: 4890
  Do not use a port number that is already assigned to another application.
- Maximum session timeout (s)
  Default value: 600000 s
- Maximum number of OPC UA sessions
  Default value: 100

Subscriptions

Define the following settings:

- Minimum publication interval (ms)
  Default value: 100 ms
- Maximum number of monitored items
  Default value: 0

Security

Secure connection

Security policies

⚠ CAUTION
Reduced security When the option "No OPC UA Server Security" is enabled, any OPC UA client can connect to the OPC UA server regardless of the following settings.

The following section contains a list of all security policies available on the server.

- Activate the required security policies.

User authentication

Guest authentication

- To allow access by anonymous users to the OPC UA server, enable the option "Enable guest authentication".
  An authentication by means of user name and password is not required for guests.
  Security is restricted to the degree that you determine by assigning rights to this user.

Authentication by means of user name and password

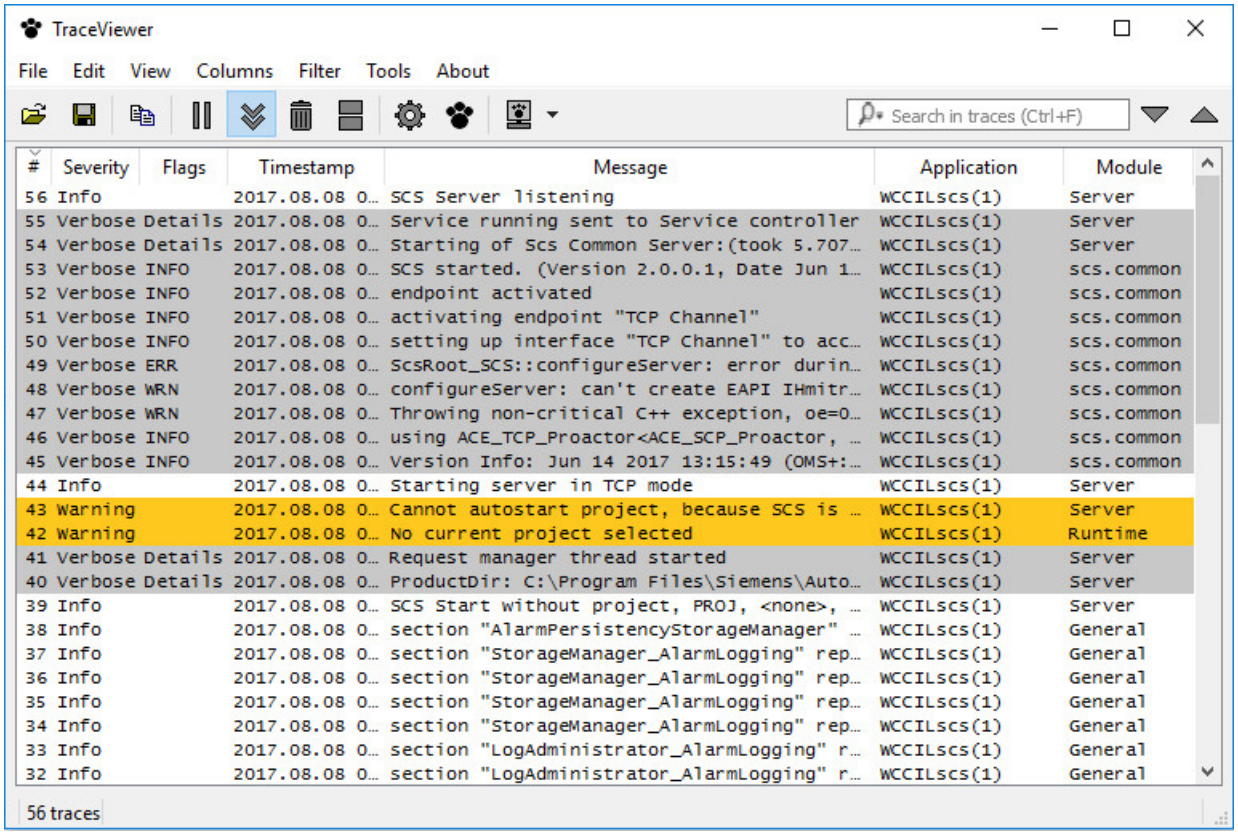
- To allow access by users with user name and password to the OPC UA server, enable the option "Authentication with user name and password".
  If access to the OPC UA server is to require the user name and password, the user must be assigned the role "HMI Administrator". The "HMI Administrator" role has the system-defined "OPC UA - read and write access" function right. The settings made must then be synchronized with the user management in runtime.

Trace

WinCC Unified provides trace logging for error analysis. The OPC UA traces including SDK can be logged for test purposes and for troubleshooting.

TraceViewer

The log files can be viewed with the TraceViewer. This tool is available in the installation directory of WinCC Unified under "WinCCUnified\bin". Start the file "RTILtraceViewer.exe".



#	Severity	Flags	Timestamp	Message	Application	Module
56	Info		2017.08.08 0…	SCS Server listening	WCCILscs(1)	Server
55	Verbose	Details	2017.08.08 0…	Service running sent to Service controller	WCCILscs(1)	Server
54	Verbose	Details	2017.08.08 0…	Starting of Scs Common Server:(took 5.707…	WCCILscs(1)	Server
53	Verbose	INFO	2017.08.08 0…	SCS started. (Version 2.0.0.1, Date Jun 1…	WCCILscs(1)	scs.common
52	Verbose	INFO	2017.08.08 0…	endpoint activated	WCCILscs(1)	scs.common
51	Verbose	INFO	2017.08.08 0…	activating endpoint "TCP Channel"	WCCILscs(1)	scs.common
50	Verbose	INFO	2017.08.08 0…	setting up interface "TCP Channel" to acc…	WCCILscs(1)	scs.common
49	Verbose	ERR	2017.08.08 0…	ScsRoot_SCS::configureServer: error durin…	WCCILscs(1)	scs.common
48	Verbose	WRN	2017.08.08 0…	configureServer: can't create EAPI IHmitr…	WCCILscs(1)	scs.common
47	Verbose	WRN	2017.08.08 0…	Throwing non-critical C++ exception, oe=0…	WCCILscs(1)	scs.common
46	Verbose	INFO	2017.08.08 0…	using ACE_TCP_Proactor<ACE_SCP_Proactor, …	WCCILscs(1)	scs.common
45	Verbose	INFO	2017.08.08 0…	Version Info: Jun 14 2017 13:15:49 (OMS+:…	WCCILscs(1)	scs.common
44	Info		2017.08.08 0…	Starting server in TCP mode	WCCILscs(1)	Server
43	Warning		2017.08.08 0…	Cannot autostart project, because SCS is …	WCCILscs(1)	Server
42	Warning		2017.08.08 0…	No current project selected	WCCILscs(1)	Runtime
41	Verbose	Details	2017.08.08 0…	Request manager thread started	WCCILscs(1)	Server
40	Verbose	Details	2017.08.08 0…	ProductDir: C:\Program Files\Siemens\Auto…	WCCILscs(1)	Server
39	Info		2017.08.08 0…	SCS Start without project, PROJ, <none>, …	WCCILscs(1)	Server
38	Info		2017.08.08 0…	section "AlarmPersistencyStorageManager" …	WCCILscs(1)	General
37	Info		2017.08.08 0…	section "StorageManager_AlarmLogging" rep…	WCCILscs(1)	General
36	Info		2017.08.08 0…	section "StorageManager_AlarmLogging" rep…	WCCILscs(1)	General
35	Info		2017.08.08 0…	section "StorageManager_AlarmLogging" rep…	WCCILscs(1)	General
34	Info		2017.08.08 0…	section "StorageManager_AlarmLogging" rep…	WCCILscs(1)	General
33	Info		2017.08.08 0…	section "LogAdministrator_AlarmLogging" r…	WCCILscs(1)	General
32	Info		2017.08.08 0…	section "LogAdministrator_AlarmLogging" r…	WCCILscs(1)	General

14.4.3.3 Using the Unified Comfort Panel as OPC UA server

Configuring the Unified OPC UA server (UCP)

OPC UA server

General

OPC is a standardized manufacturer-independent software interface for data exchange in automation engineering. OPC UA is the technology succeeding OPC. OPC UA is platform-independent and supports different protocols as communication medium.

To work with OPC UA in WinCC Unified, the OPC UA server must be enabled in the TIA Portal in the Runtime settings of the HMI device.

Read/write tags and register tags/alarms

When you enable the "Operate as OPC UA server" option in the HMI device, the protection for unauthorized internal and external access is downgraded.

- Enable the "Operate as OPC UA server" option.
  A security note is displayed.

After enabling the option, all other settings of the OPC UA Server will become available.

Alarms and Conditions

- To display alarm conditions in the address range of the server, select the option "Enable Alarms and Conditions on the OPC UA server".
- To disable or acknowledge alarms on the OPC UA Client, for example, select the option "Allow operation of alarms on the OPC UA Client". To enable this option, the "Enable Alarms and Conditions on the OPC UA server" option must be enabled.

Options

General

Define the following settings:

- Port
  Default value: 4890
  Do not use a port number that is already assigned to another application.
- Maximum session timeout (s)
  Default value: 600000 s
- Maximum number of OPC UA sessions
  Default value: 100

Subscriptions

Define the following settings:

- Minimum publication interval (ms)
  Default value: 100 ms
- Maximum number of monitored items
  Default value: 0

Security

Secure connection

Security policies

⚠ CAUTION
Reduced security When the option "No OPC UA Server Security" is enabled, any OPC UA client can connect to the OPC UA server regardless of the following settings.

The following section contains a list of all security policies available on the server.

- Activate the required security policies.

User authentication

Guest authentication

- To allow access by anonymous users to the OPC UA server, enable the option "Enable guest authentication".
  An authentication by means of user name and password is not required.
  Security is restricted to the degree that you determine by assigning rights to this user.

Authentication by means of user name and password

- To allow access by users with user name and password to the OPC UA server, enable the option "Authentication with user name and password".
  If access to the OPC UA server is to require the user name and password, the user must be assigned the role "HMI Administrator". The "HMI Administrator" role has the system-defined "OPC UA - read and write access" function right. The settings made must then be synchronized with the user management in runtime.

14.4.4 WinCC Unified OPC UA client

14.4.4.1 Using the WinCC Unified OPC UA client

As OPC UA clients, WinCC Unified devices can integrate the following data from OPC UA servers into their projects:

- Alarm instances received from the OPC UA server
- OPC UA server tags

When configuring this data in the engineering system, the engineering system also becomes the OPC UA client.

14.4.4.2 Defining connection settings to the OPC UA server

Requirement

- In the engineering system, a WinCC project is open that has had a Unified device added to it.
- The "Connections" editor is open.

Procedure

Double-click in the "Add" cell and define the connection settings:

- "Communication driver": OPC UA
- Set the following parameters in the "Parameters" tab under "OPC server":
  - "UA server discovery URL": Enter the OPC UA server IP and port
    Use the following notation: opc.tcp://<IP>:<Port>
    Alternatively, you can also determine the server via "Select OPC server".
  - Select the desired security settings.
    See also Defining the security settings for communication with the OPC UA server (Page 7045).

Result

The Unified OPC UA client uses the settings to establish a connection to the OPC UA server.

14.4.4.3 Defining the security settings for communication with the OPC UA server

Requirement

A connection with the communication driver "OPC UA" is configured in a WinCC project on a Unified device. See also section Defining connection settings to the OPC UA server (Page 7045).

Procedure

Select the security settings that meet the requirements of the OPC UA server:

1. Open the "Connections" editor.
   You make the security settings in the "OPC UA Server" area.
2. To protect the connection with a security policy, follow these steps:
   – Select the security policy.
   The communication with the server is protected by a certificate.

   Note

   Make OPC UA certificates available

   Make sure that the required certificates are available on the OPC UA server and client. See also Introduction to OPC UA certificates (Page 7012).

   Note

   Connection without security policy

   If you do not select a security policy, it is urgently recommended that the OPC UA server and client are installed on the same device.

   – Select whether communication is signed or signed and encrypted.
3. To protect communication with the OPC UA server by a user name and password, follow these steps:
   – Disable the "Anonymous" option.
   – Enter the user name and password of a user account configured on the OPC UA server.
4. For anonymous communication, select the option "Anonymous".

14.4.4.4 Integrating OPC UA server alarm instances into a Unified client

You have the option of integrating alarm instances from an OPC UA server into your Runtime project.

Note

Restrictions

- The OPC UA server is a SINUMERIK device.
- The OPC UA server is based on the OPC UA specification 1.03.

Requirement

- The OPC UA server alarm instances are available in a NodeSet XML file.
- You have access to the XML file on the device on which the engineering system is installed.

Procedure

1. In the engineering system, add a Unified HMI device to a WinCC project.
2. Set the connection settings to an OPC UA server for the HMI device.
   See also section Defining connection settings to the OPC UA server (Page 7045).
3. Import the XML file with the OPC UA server alarm instances into the WinCC project.
   See also section Importing OPC UA server alarm instances (Page 7047).
4. Generate HMI alarms for the OPC UA server alarm instances.
   See also section Generating HMI alarms for OPC UA server alarm instances (Page 7048).
5. Add a screen to the HMI device.
6. Place an alarm display on the screen.
7. Compile the HMI device in a Runtime project, load the Runtime project onto the HMI device and start the project in Runtime.

Note

Loss of the ability to compile and load changes

If you load the OPC UA server alarm instances on the HMI device and then update the alarm instances in the engineering, because the alarm class has been changed for example, you lose the option in the engineering to compile and load only the changes to the project. The project must now be fully compiled and loaded.

Result

In Runtime, the alarm instances received from the OPC UA server are displayed in the alarm display. The following attributes are mapped to each other:

Attributes of an OPC UA server alarm instance	Attribute of an HMI alarm
"Message"	"Alarm text"
"SourceName"	"Origin"

If alarm archiving is activated, the alarms are archived.

Status changes to the alarm instances on the OPC UA server are reflected in Runtime. The Runtime OPC UA client can request status changes on the server. The status change is always done on the server.

Importing OPC UA server alarm instances

Requirement

- In the engineering system, a WinCC project is open that has had a Unified device added to it.
- The OPC UA server alarm instances are available in a NodeSet XML file.
- You have access to the XML file on the device on which the engineering system is installed.

Procedure

To import OPC UA server alarm instances into a WinCC project, follow these steps:

1. Open the "HMI alarms" editor.
2. Select the "OPC UA A&C" tab.
3. Under "Connection" in the right area, select the OPC UA connection.
4. Click "Import" next to "Connection":

5. Select the XML file.
6. Click "Import".

Result

The content of the XML file is imported into the "OPC UA browser" area. It contains the hierarchical OPC UA NodeSpace with the OPC UA server alarm instances.

Then generate HMI alarms for the OPC UA server alarm instances.

See also

Generating HMI alarms for OPC UA server alarm instances (Page 7048)

Generating HMI alarms for OPC UA server alarm instances

Requirement

- In the engineering system a WinCC-project is open, to which a HMI device has been added.
- The "HMI alarms" editor of the device is open.
- An XML file with the OPC UA server alarm instances was imported into the editor.

Procedure

1. Select the "OPC UA A&C" tab.
2. Expand the objects in the "OPC UA browser" area up to the node under which the OPC UA server alarm instances are located.
3. Press and hold the left mouse button to drag the node to the "Node ID" cell of the "Add" row of the table in the "OPC UA types" area.
   An entry for an alarm type is added to the table. The table provides detailed information on the properties of the alarm type.

4. Select an alarm class that matches the alarm type of the OPC UA server alarm instances and supports the state machine "Alarm without status active with acknowledgment".
5. In the "OPC UA types" area, click the button to generate and update the alarm instances:

Note

Loss of the ability to compile and load changes

If you load the OPC UA server alarm instances on the HMI device and then update the alarm instances in the engineering, because the alarm class has been changed for example, you lose the option in the engineering to compile and load only the changes to the project. The project must now be fully compiled and loaded.

Result

HMI alarms are generated for the OPC UA server alarm instances.

When loading the project into a Runtime, the mapping between the HMI alarms and the OPC UA server alarm instances are loaded into the target device.

See also

Importing OPC UA server alarm instances (Page 7047)

Configuring plant hierarchies 15

15.1 Basics

15.1.1 Introduction

Object-oriented configuration

The option of object-oriented configuration is available to you with SIMATIC WinCC Unified PC RT. Define reusable plant object types and assign the associated plant object instances in hierarchical plant views.

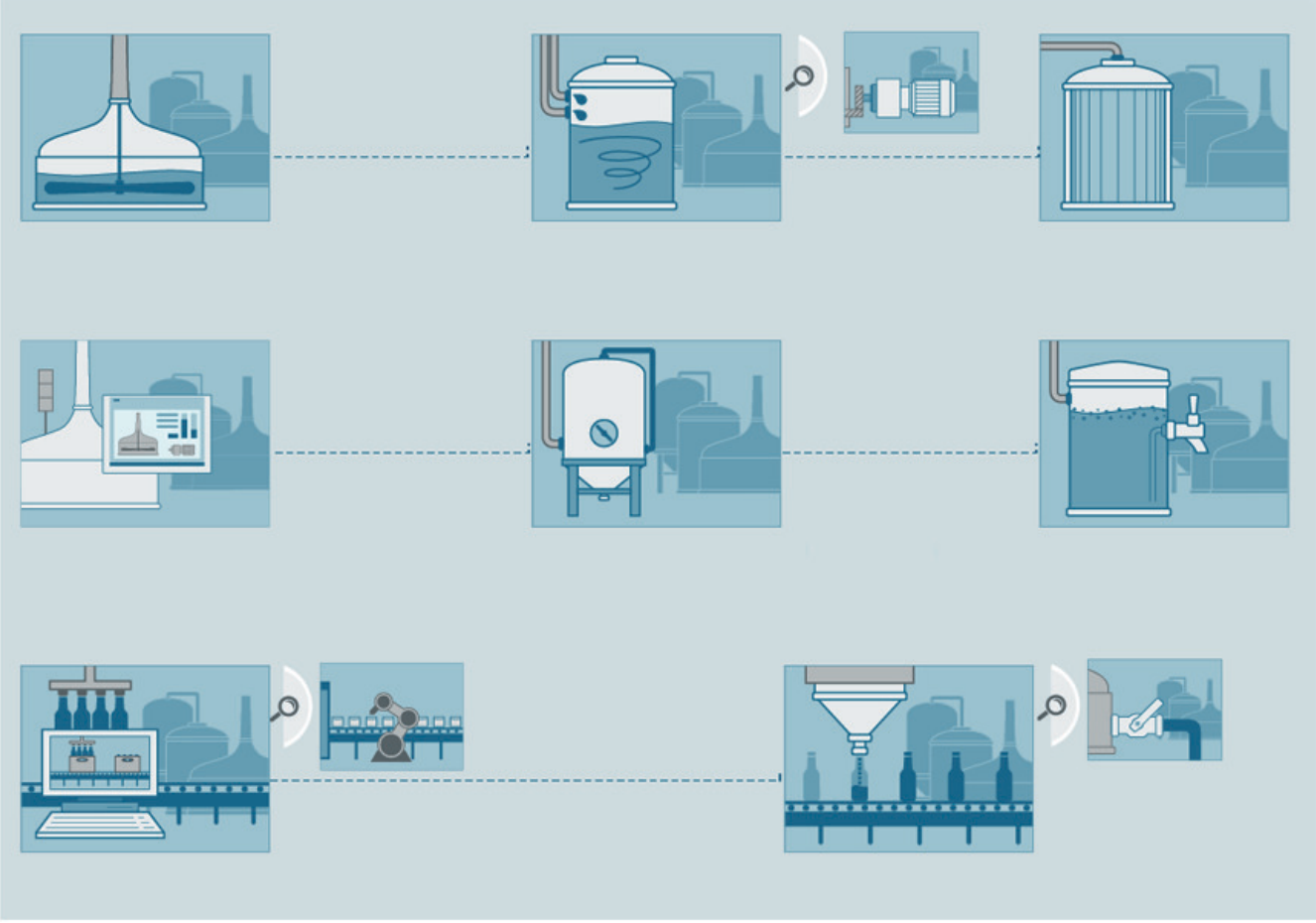
In this way, you can model the plant view of your machine or unit/plant, for example, based on user-defined or standardized technological objects.

The plant structure is created from individual objects, each of which represents a specific component or unit. You configure each object in the context of the operator control and monitoring solution.

In plant object types, you combine all required configuration elements for visualization, such as faceplates, tags, alarms, scripts, etc. Changes to the plant object type automatically affect all instances. This translates into real time savings, especially for plants with a high degree of standardization.

You can start object-oriented plant modeling based on the engineering data, if necessary, and can derive the configuration of the HMI devices and automation systems from this.

Break the machine or unit/plant up into reusable technological units and arrange them hierarchically in a technological plant view according to the plant structure.



The following options are available to you in technology-oriented and object-oriented configuration:

- Creating various hierarchical plant views: technological view, building view, independent of the HMI device that is used.
- Configuration of plant objects and plant object types with data elements for mapping the actual plant configuration
- Access to plant objects (data elements, HMI alarms, logs, screens, etc.)
- Generation of the screen hierarchy
- Expansion of configured plant objects and types using Plant Intelligence options, e.g. WinCC Unified Performance Insight or WinCC Unified Sequence Execution

Software requirements

You acquire the following products to use technology- and object-oriented configuration:

- TIA Portal V16 or higher with WinCC Unified

The "Plant objects" area is visible under Project tree after the installation of WinCC Unified.

Supported devices

The following SIMATIC S7 controllers are supported:

- SIMATIC S7-1500 with the exception of the SIMATIC S7-1500 Software Controller

See also

Applications (Page 7053)

Overview (Page 7065)

Type/instance concept in object-oriented configuration (Page 7055)

15.1.2 Applications

Overview

You use technology-oriented and object-oriented configuration for automation solutions to increase overall effectiveness.

In particular, in plants with high level of standardization, the object-oriented approach increases the configuration efficiency through the reuse of objects, the capability of changing objects centrally, and the integration of manufacturing execution system functionalities such as the calculation of individual key indicators for separate machines.

Technology-oriented and object-oriented configuration supports you in the following operating phases:

- Planning phase: Efficient plant configuration and simple plant expansion through integration of part models from other projects
- Plant maintenance: Transparency through mapping of the exact plant structure
- Quality management: Continuous optimization of your projects

Advantages

- Creation and generation of modular projects based on standardized plant objects
- Reduced engineering workloads and fewer inconsistencies with a shared model in Engineering and Runtime
- Simple plant expansion through integration of part models from other projects
- Creating the screen hierarchy
- Transparent access to all objects and their properties and methods, independent of device assignment
- Targeted corrective measures through transparent relationships of individual plant objects
- Intelligent use of information from the entire manufacturing environment in combination with Plant Intelligence options

Operation in runtime

Depending on your configuration, the following possibilities are available to you in runtime:

- Display hierarchy path of alarm source
- Filter alarm control by plant objects
- Display alarm status of a line and navigation to the alarm source
- Display the most frequently occurring alarms, filtered by plant object or plant object type
- Area-based access protection
- Screen navigation via the plant model
- Determine the energy consumption of a line and compare with another line
- Analysis based on plant objects

The plant hierarchy is also available for scripting in runtime.

Requirements on the configuration engineer

The following prior knowledge is required for using technology-oriented and object-oriented configuration:

- You have experience performing configuration in STEP 7 and WinCC.

See also

Introduction (Page 7051)

Type/instance concept in object-oriented configuration (Page 7055)

Configuration concept (Page 7058)

15.1.3 Type/instance concept in object-oriented configuration

Introduction

The object-oriented approach of WinCC based on the type-instance concept. In types, you create central properties for an object. The instances represent local places of use for the types.

Plant objects are instances of a plant object type.

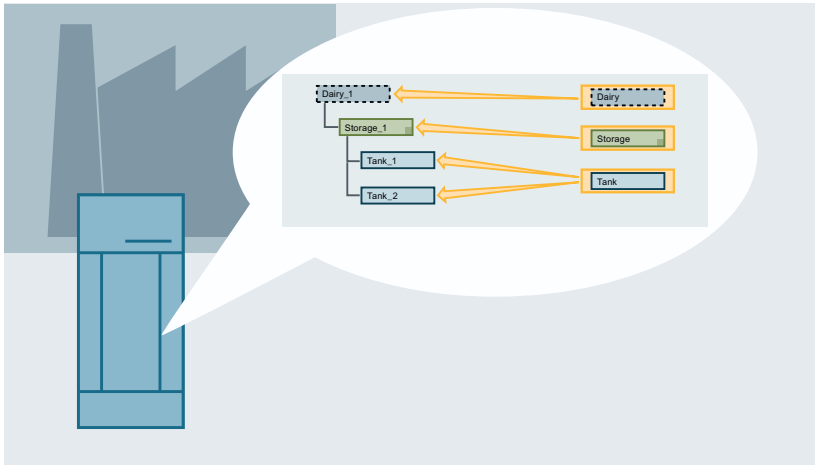
The plant object type is the central, object-oriented definition of a reusable plant component (such as conveyor robot). As instances of the plant object type, the plant objects generally map concrete, physically existing plant components (e.g. conveyor robot_A and conveyor robot_B).

If you change a property of a plant object type, the property is saved centrally and also changed in all instances.

Effect of the type instance concept on object-oriented configuration

The use of a type is called an instance. The common plant model is generated from instances. Each instance inherits all the properties of the type. The Common Plant Model with high level of standardization is characterized by the use of many instances of few types in the model.

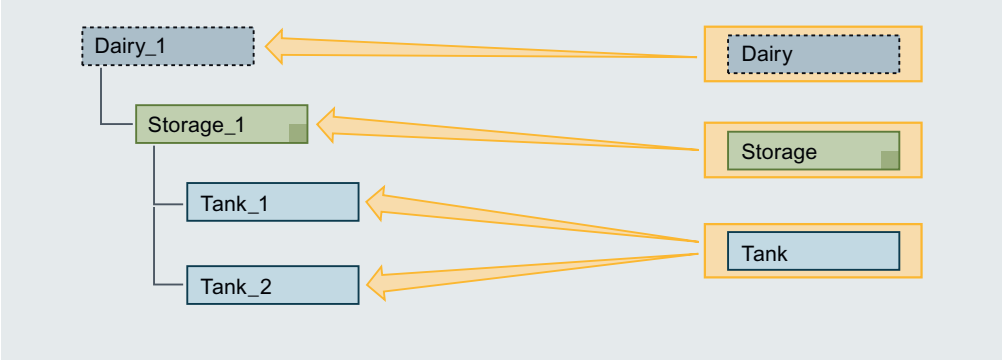
The general types of the plant units are configured and these are reused when required in the configuration and adapted to the specific plant objects. The plant structure hereby specifies the addressing of the plant objects.



In the object-oriented approach of WinCC the following correspondences apply:

- Type = Plant object type
- Instance = Plant object

The following figure shows the basic structure of a plant model:



Plant objects and plant object types

A plant object is a technological unit. In a plant object, the components are stored in a typical form which is required for modeling a plant.

A valid plant object must be created from a plant object type. The plant structure is created from plant objects.

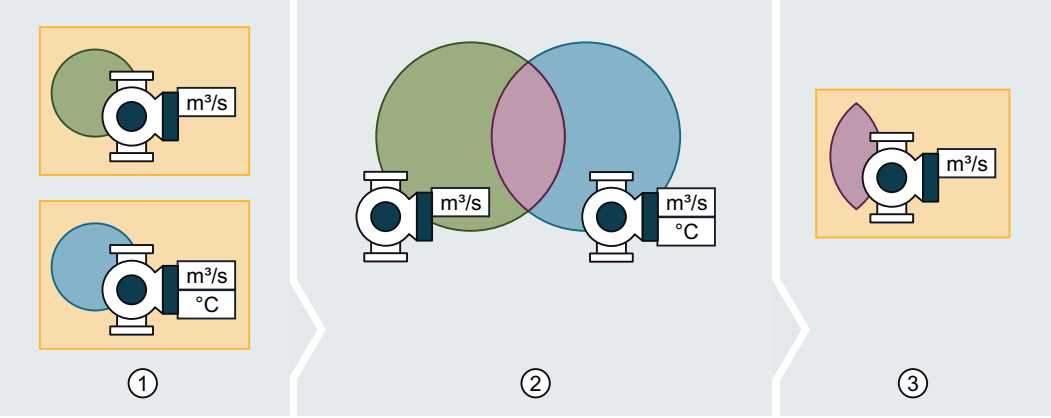
The definition of a plant object type consists of the data structure and context information:

- Alarms
- Logging
- Visualization
- Data member (internal and external)
- Facets (e.g. performance indicators)

Type definition in terms of high reuse

A plant object type is used to describe a plant object independently of its use in the Common Plant Model. Define a plant object type as generally as possible and as specifically as required. Take into account the following aspects:

- Identical data structure in PLC (function block or PLC UDT)
  Example: Pumps that have different performance ranges are installed in a plant. The data structure in the PLC is identical for each pump. Map these pumps with a common plant object type. At each instance you configure the specific value ranges for the respective performance ranges.
  A pump function block (standard FB for a pump) is available on the control side. The customer defines the plant object type "Pump" based on this function block. The data structure of the plant object type is taken over directly from the block. Only the HMI relevant properties from the function block are hereby transferred. They are automatically updated when the block changes. Simply parameterize an instance of the function block as process connection of the plant objects.
- Similarity
  When you have similar plant object types, check if it possible to map these with a common plant object type:



① Example: A pump is installed in a plant in two different variants:
- Variant 1 only measures the flow rate.
- Variant 2 measures the temperature in addition to the flow rate.

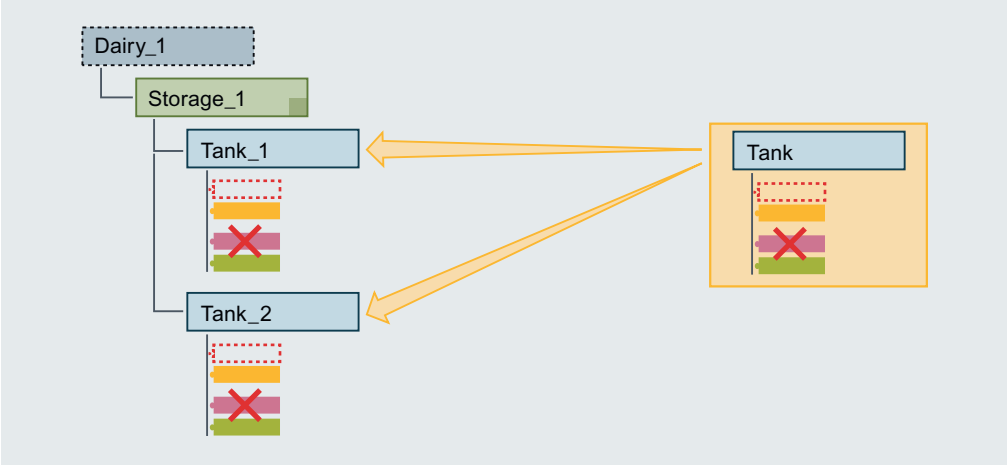
Effects on the definition of the plant object types:
- You map each of the two pumps with a single plant object type. The representation in the Common Plant Model hereby corresponds to reality.
- There is more configuration work.

② The common intersection of the two pumps is measuring the flow.

③ If, for example, you can do without measuring the temperature for operation, define only one plant object type:
- There is less configuration work.
- The two variants of the pumps are not fully represented in the Common Plant Model.

Effects of changes on plant object types

The following figure shows how changes to the plant object type affect its instances, i.e. plant objects:



See also

Overview (Page 7065)

Options for creating plant objects (Page 7068)

Introduction (Page 7051)

Applications (Page 7053)

Configuration concept (Page 7058)

Plant model and target systems (Page 7060)

15.1.4 Configuration concept

Requirements

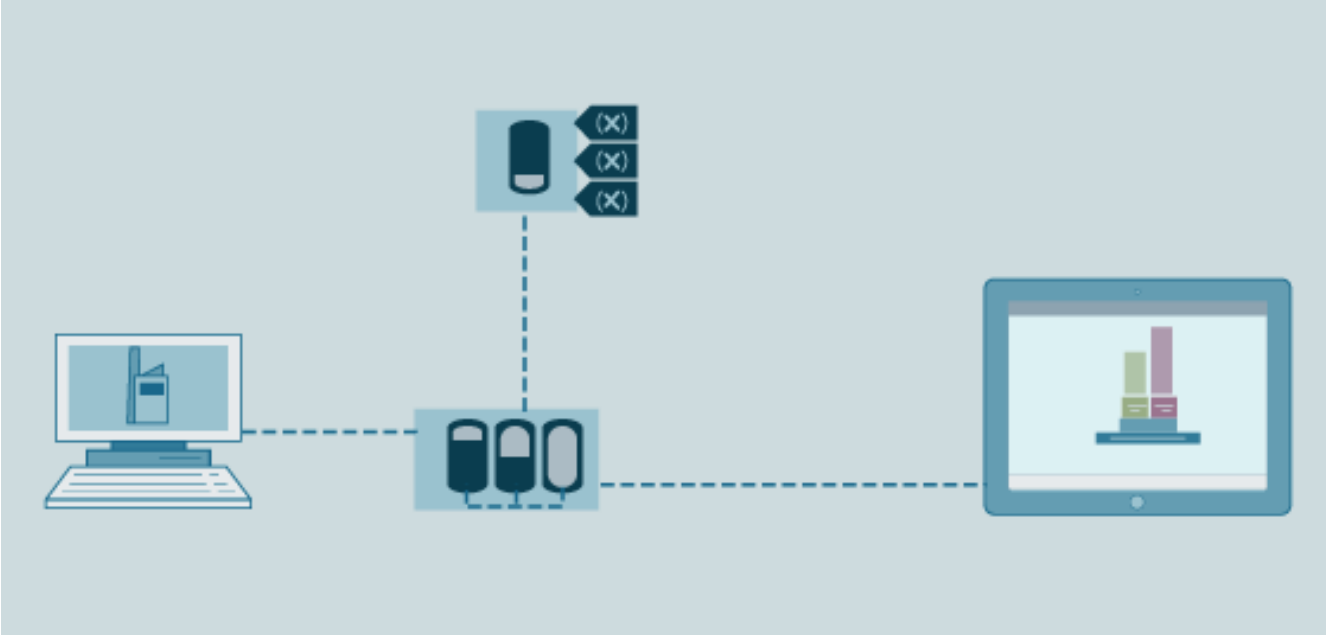
- You have experience in configuring with WinCC and STEP 7.
- The TIA Portal project has been created.
- The WinCC Unified PC RT HMI device has been created.
- A SIMATIC S7-1500 PLC has been created.
- Data blocks are configured in the PLC.

Workflow for configuration

The starting point for the definition of a standardized object-oriented plant model in object oriented configuration is the existing plant structure.

If you want to create a plant structure, use the following sequence of steps as a guide:

- Analyze the plant structure and break it down into units and components (plant objects)
- Identify required plant object types
- Define data of the plant object types based on FBs and PLC UDTs
- Define hierarchical plant view using instances
- Create a target system
- Map the plant structure
- Position plant objects in the plant structure
- Add functional facets to object types, e.g. assign shift calendars for all machines of a line or plant



Tips for an efficient procedure

If you are using pre-planning and automation engineering tools, you can have your plant structure automatically created via TIA Portal Openness. Next, set up the process connection of the plant objects via TIA Portal Openness.

Differences between device-oriented and object-oriented configuration

In technology- and object-oriented configuration, you work with objects with relevant names instead of individual tags or faceplates, for example.

You have access to all objects and their properties, methods, etc. in the hierarchy, independent of HMI device assignment.

The equipment from different products and versions is integrated in the object-oriented configuration.

Using multiuser engineering

If you use multiuser engineering in object oriented configuration, you can save your changes only in the server project view. You cannot check the changes you make in the local session into the server project.

You can find more information on Multiuser Engineering in "Using Multiuser Engineering".

See also

Creating plant objects (Page 7072)

Structure of a plant model (Page 7061)

Creating plant object types (Page 7071)

Configure plant object types (Page 7073)

Creating a plant hierarchy (Page 7069)

Type/instance concept in object-oriented configuration (Page 7055)

Plant model and target systems (Page 7060)

Applications (Page 7053)

15.1.5 Plant model and target systems

Configuration of the plant model

When the configuration of a visualization solution begins, the development of the automation solution often takes place in the final phase. Initially, only the actual plant structure is relevant for mapping the plant model. Whether this involves one or multiple target systems is initially irrelevant.

There are always two views in a WinCC project:

- Device view with configured target systems
- Object-oriented view (common plant model)

You can perform configuration independently in both views.

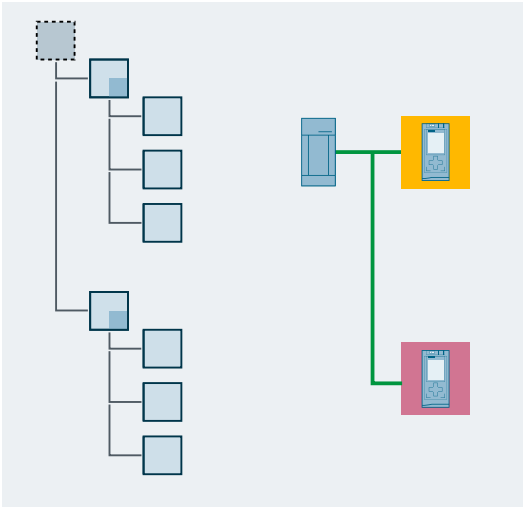
Process connection of the plant model

The target systems are the interface between the common plant model and the process. One or more connections to PLCs are configured on each target system. The plant objects communicate with the PLCs over the target systems.

Your project must meet the following conditions for productive use:

- Each plant view is assigned to a HMI device.
- Each plant object with a process connection is also connected to a PLC.

The following figure shows a schematic representation of the mapping of plant objects to the configured target systems and PLCs:



Plant objects without a process connection as representation of a unit

Plant objects with process connection

Runtime server (target system)

PLC

See also

Type/instance concept in object-oriented configuration (Page 7055)

Configuration concept (Page 7058)

15.1.6 Structure of a plant model

Basic principles

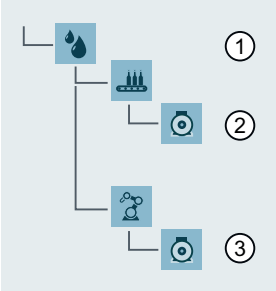
With object oriented configuration, a configured plant object corresponds to a real plant object. Basically, the number of plant objects is determined by the plant hierarchy.

Whether you need to map each plant object with a plant object type is determined by the following factors:

- Relevance of the plant object type for the process visualization
- Depth of the plant hierarchy that is to be mapped
- Degree of reuse

The specific function of a plant object is clear from its position in the plant hierarchy.

For example, the function of a "Drive" plant object is only revealed in the plant hierarchy:

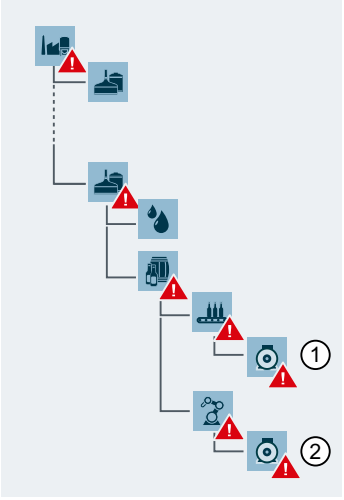


① Process for filling beer into bottles
② Drive for conveyor belt
③ Drive for robot

Depth of the plant hierarchy

Define any depth of the plant hierarchy The depth of the plant hierarchy depends essentially on the number of plant objects. A deep plant hierarchy leads to a precise fault localization. You can then, for example, formulate the concise alarm text.

The context of the plant object is also taken into account in runtime, for example, in the localization of faults. The following figure uses the example of the "Temperature exceeded" alarm to show the advantage a deep hierarchy offers in runtime:



① Representation of the message in an alarm control:
"Brewery.Filling.Paletting.Robots.Drive.Temperature exceeded"
The Common Plant Model with deeper hierarchy leads to a precise fault localization. You can therefore formulate the alarm text concisely.

② Since the drive for the robot is based on the same plant object type, the context of the alarm is automatically correct when a fault occurs:
"Brewery.Filling.Paletting.Robots.Drive.Temperature exceeded"

Configuration data at the plant object type

The following configuration data are created during the definition of a plant object type:

- Properties through which data is exchanged inside and outside of WinCC Unified PC RT.
- HMI visualization: Alarms, logs
- KPIs

See also

Configure plant object types (Page 7073)

Configuration concept (Page 7058)

15.1.7 Contexts

Contexts allow you to view plant units according to a certain viewpoint, e.g. according to a certain customer, product, job or shift.

Principle

Contexts always belong to a plant object. They are indicated as follows:

- User-defined contexts:
  Using a program created with the ODK API
- System-generated contexts:
  For installed Performance Insight and Calendar option packages: Automatically in Runtime
  Example: When a shift starts in Calendar, an archived context value is created with the shift ID

Each time a context (e.g. "Product") is executed, a log entry is generated in the context log. The logged context saves:

- The context value (e.g. "orange lemonade")
- Start time and end time of the execution time
- The quality code

Contexts in the trend control and alarm control

You can filter the content of these controls so that only data that has been generated in a specific plant unit and for the context you have selected is displayed. To do this, select a plant object, a context and one of its logged context values.

Example:

A press house produces juices for various beverage brands. Using contexts, employees can display in Runtime which alarms have occurred:

- During the production of a specific product (e.g. natural apple juice, clear apple juice, pear juice)
- For orders for a specific customer (e.g. Johnson, Smith or Miller).
- During a specific shift (e.g. early shift, late shift, night shift).

Contexts in the "Reports" control

You have the option of linking the generation of reports to the execution of contexts.

If the templates are configured appropriately, the reports available in the control can also contain information about contexts. If a report was generated as an Excel file and reads both contexts and alarms or tag values, you can then use the Excel filter function to filter the alarms and tags by context.

See also

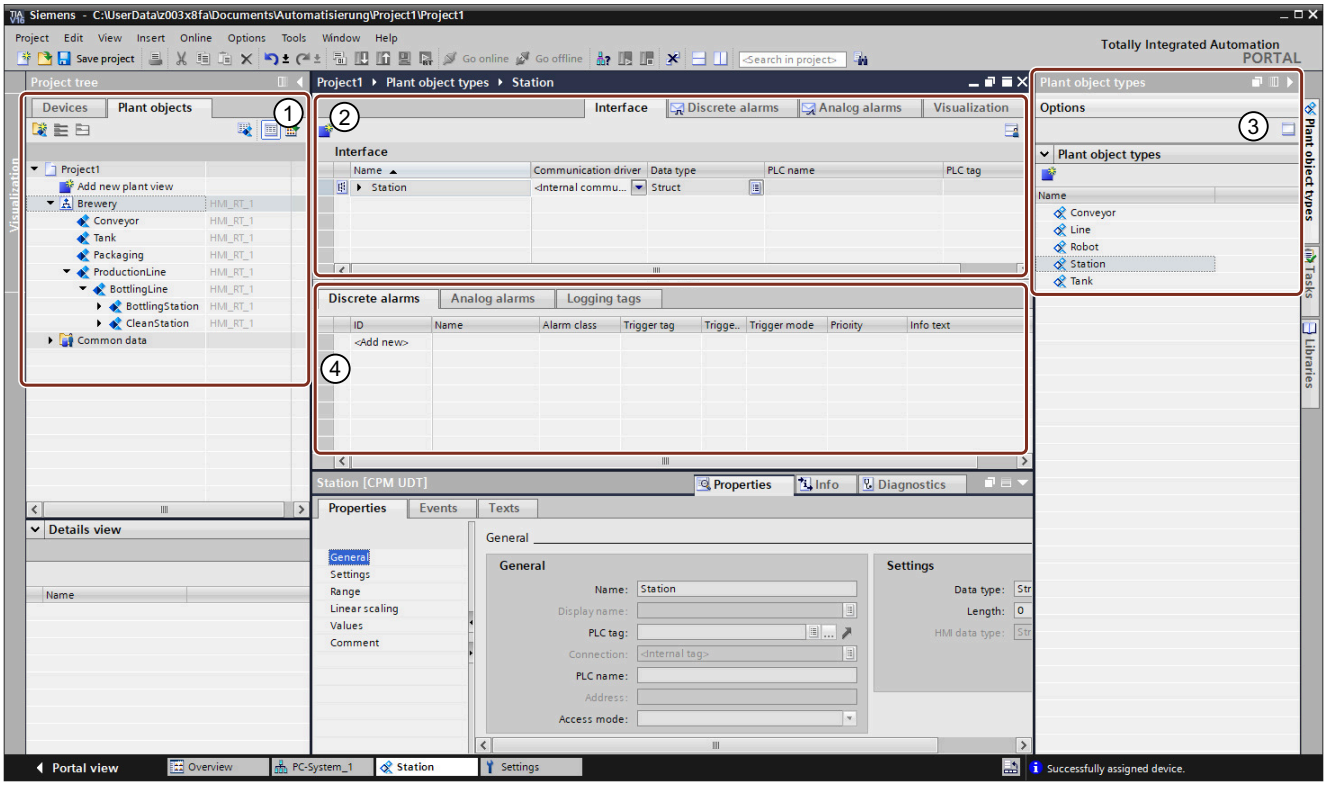
Display process data of the plant objects in a trend control (Page 7116)

15.2 Elements and basic settings

15.2.1 Overview

"Plant objects" area

To access object-oriented configuration, click on "Plant objects" in "Project tree".



① "Plant objects" area for object-oriented configuration

② Plant object specific tabs, e.g. "Interface", "Visualization", etc.

③ "Plant object types" task card

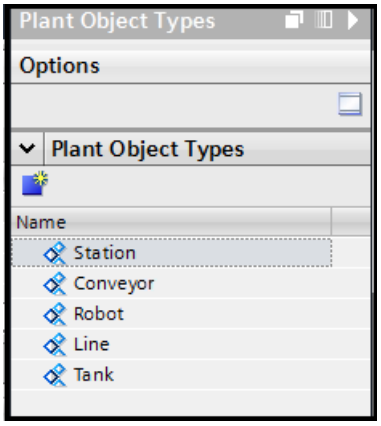
④ Tabs for the configuration of alarms and logs for plant objects

Create the plant view under "Project tree > Plant objects". You can create a plant view in a project. The plant view is filled with plant nodes and thus maps your plant. Plant nodes act as structural elements. Create plant objects based on the plant object types created in the project.

In the "Plant objects" area you assign an HMI device to the plant view.

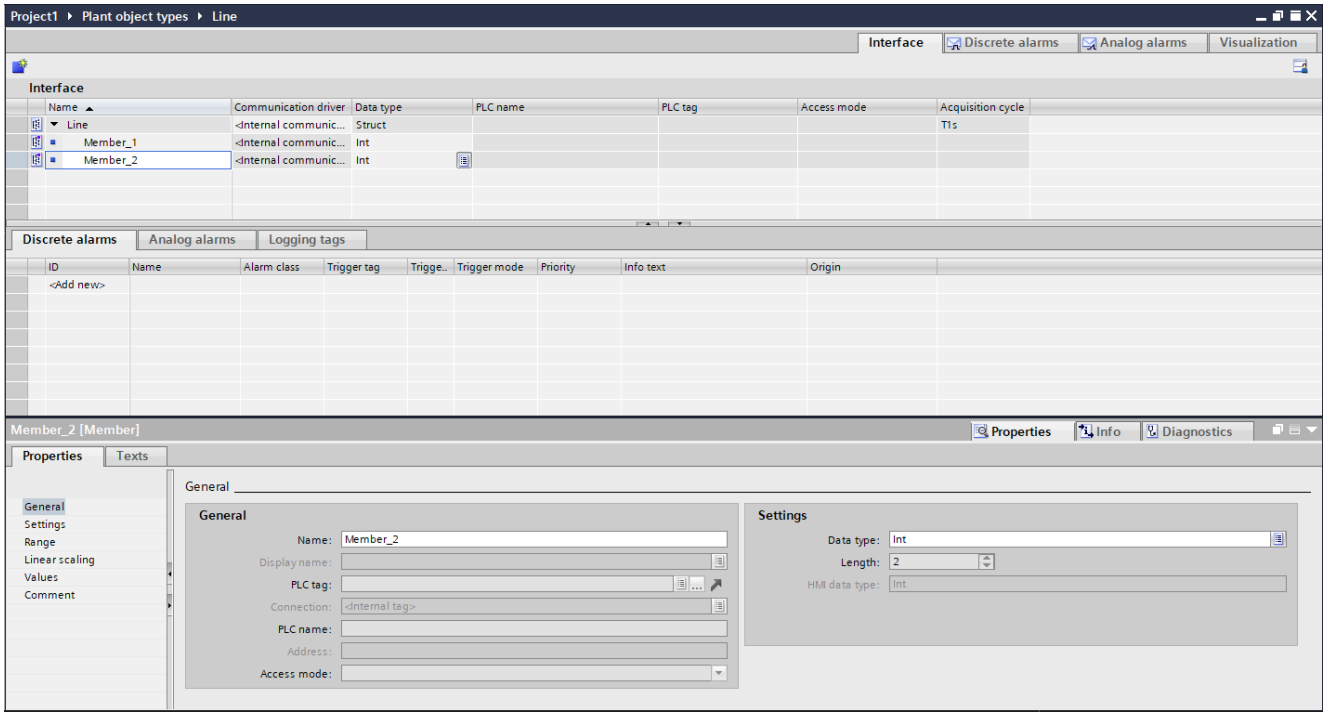
"Plant object types" task card

Under "Plant object types", create the plant object types from which you create plant objects.



"Interface" tab

Plant object types are edited in "Interface" create tags for the communication between a PLC and an HMI device, create members for plant object types, and create alarms and logging tags.



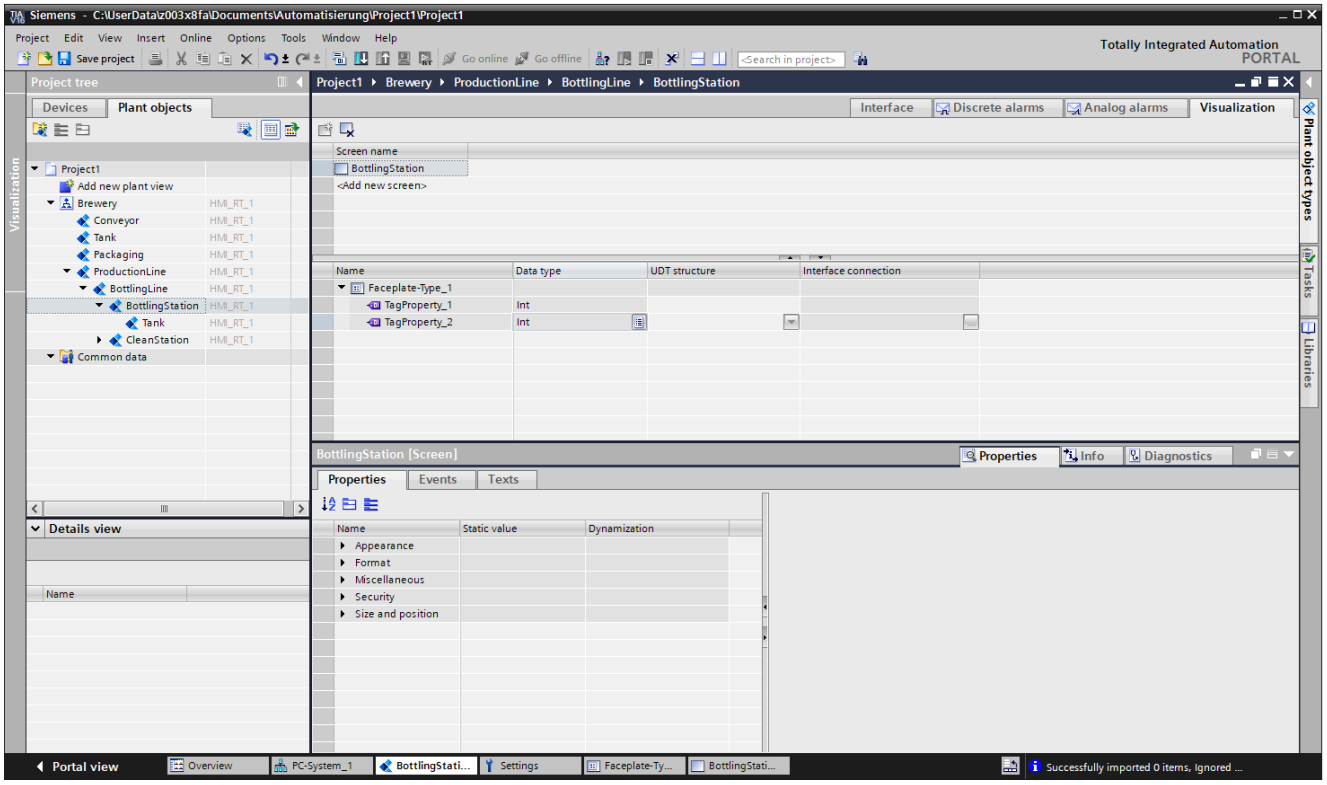
"Visualization" tab of the plant object types

In the "Visualization" tab of a plant object type, you link a faceplate type with the plant object type.

"Visualization" tab of the plant objects

Under "Visualization", you configure a screen for each plant object. In the Inspector window you edit the properties and events of the screen.

The faceplate type associated with the plant type is displayed. The configured tags of the interface are displayed, but cannot be edited.



If you open a screen under "Visualization" by double-clicking, the view is identical to the view on an HMI device. The "Toolbox" and "Layout" task cards are also identical.

Use the "Toolbox" task card to configure in predefined objects in your screens, with which you map your plant, display process sequences and define process values.

See also

Introduction (Page 7051)

Options for creating plant objects (Page 7068)

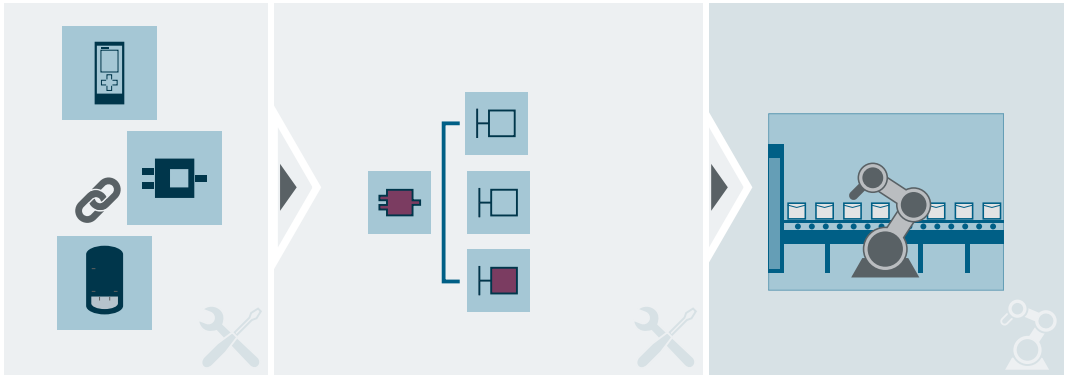
Type/instance concept in object-oriented configuration (Page 7055)

15.2.2 Options for creating plant objects

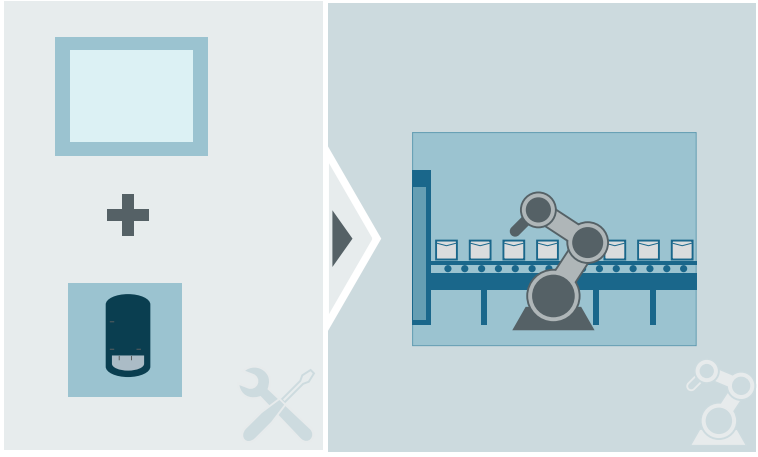
Basics

You have several options for creating plant objects on the basis of plant object types:

- Creation of plant object types from the function blocks or PLC user data types of an S7-1500 and creation of plant objects from the instance data blocks



- Creation of plant object types within WinCC without an S7-1500



See also

Overview (Page 7065)

Type/instance concept in object-oriented configuration (Page 7055)

15.3 Object- and technology-oriented configuration

15.3.1 Working with plant views

15.3.1.1 Creating a plant hierarchy

Introduction

Create a plant view to map the structure of your plant. You fill the plant view with plant objects and plant nodes and thus map your plant. Plant nodes act as structural elements.

Assign the plant view to a HMI device.

Requirement

- The TIA Portal project has been created.

Procedure

1. Under "Project tree > Plant objects", click on "Add new plant view".
   An empty plant view is created.

   Note

   A plant view is supported in each project.

2. Rename the plant view accordingly.

   Note

   The following options are not available for the "Plant view" object:
   - Paste
   - Cut
   - Drag-and-drop

See also

Assigning a plant hierarchy to a HMI device (Page 7070)

Creating plant objects (Page 7072)

Configure plant object types (Page 7073)

Configuration concept (Page 7058)

15.3.1.2 Assigning a plant hierarchy to a HMI device

Introduction

To operate the plant in runtime, always assign a plant view to an HMI device.

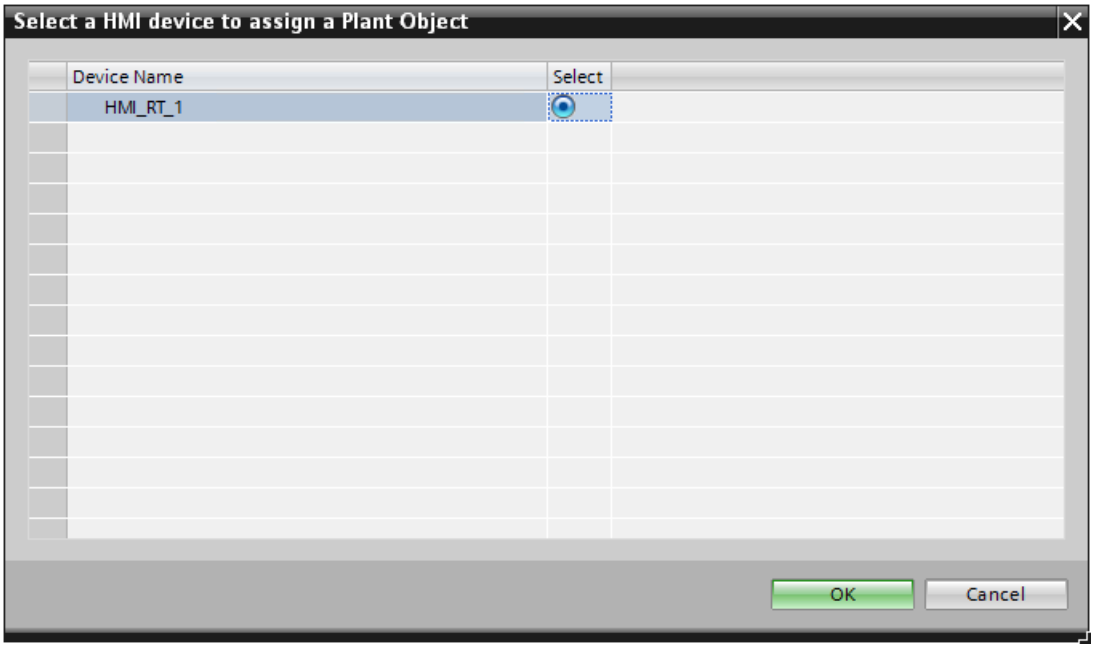
A plant view can only be assigned to a HMI device.

Requirement

- A plant view has been created.
- The WinCC Unified PC RT HMI device has been created.

Procedure

1. Select the "Plant view" node.
2. Select the "Assign HMI device" entry from the shortcut menu.
   A "Select an HMI device for assignment" dialog appears.



3. Select the HMI device.
   The plant view and all lower-level plant objects are assigned to the HMI device.
   If a plant view was assigned to a HMI device, the assignment is visible under "Project tree > Plant objects".

See also

Creating plant objects (Page 7072)

Creating plant object types (Page 7071)

Configure plant object types (Page 7073)

Creating a plant hierarchy (Page 7069)

15.3.1.3 Creating plant nodes

Introduction

Plant nodes help you structure your plant.

Requirement

- The plant view has been created and is displayed.

Procedure

1. Open the shortcut menu in the plant view.
2. Select "Add new node".
   The plant node is created.
3. Rename the plant node.

15.3.2 Working with plant objects and plant object types

15.3.2.1 Creating plant object types

Introduction

You create plant object types.

Then define the "Communications driver" property of the interface:

- "<Internal communication>": Create data members for internal communication.
- "SIMATIC S7 1200/1500": Use either function blocks or the PLC user data types of an S7-1500. You can add further data members to the linked structure.

Requirement

- A project is open.
- A SIMATIC S7-1500 PLC has been created.

Procedure

Create plant object types in the "Plant object types" task card.

1. To display the "Plant object types" task card, click the "Show plant object types" button under "Project tree > Plant objects".
2. To create a plant object type, click "Add new plant object type".
   An empty plant object type is created.
3. Rename the created plant object type accordingly.
4. To edit the plant object type or create lower-level objects and members for the plant object type, double-click the plant object type in the "Plant object types" tab.
   The plant object type appears under "Interface".

Note

The "Communications driver" property is editable for the plant object types. The property "PLC tag" can only be edited with the communications driver "SIMATIC S7 1200/1500".

See also

Configure plant object types (Page 7073)

Example: Determine plant object type (Page 7098)

Assigning a plant hierarchy to a HMI device (Page 7070)

Configuration concept (Page 7058)

15.3.2.2 Creating plant objects

Introduction

You create plant objects from a plant object type using drag-and-drop operation.

Plant objects are specific versions or instances of a plant object type.

Requirement

- A project is open.
- A plant object type has been created.

Procedure

1. Open the "Plant objects" tab in the "Project tree" area.
2. Open the "Plant object types" task card.

3. Drag the plant object type from the task card to the plant view.
   An empty plant object is created.
4. Rename the plant object accordingly.

Note

The name of a plant object must only be assigned once within a project.

See also

Configuration concept (Page 7058)

Configure plant object types (Page 7073)

Assigning a plant hierarchy to a HMI device (Page 7070)

Creating a plant hierarchy (Page 7069)

15.3.2.3 Configure plant object types

Introduction

Configure the plant object types either from the function blocks and PLC user data types of an S7-1500 or create the properties and the external and internal data members for the plant object types.

In both cases you can extend the structure the created plant object types with additional internal or external data members.

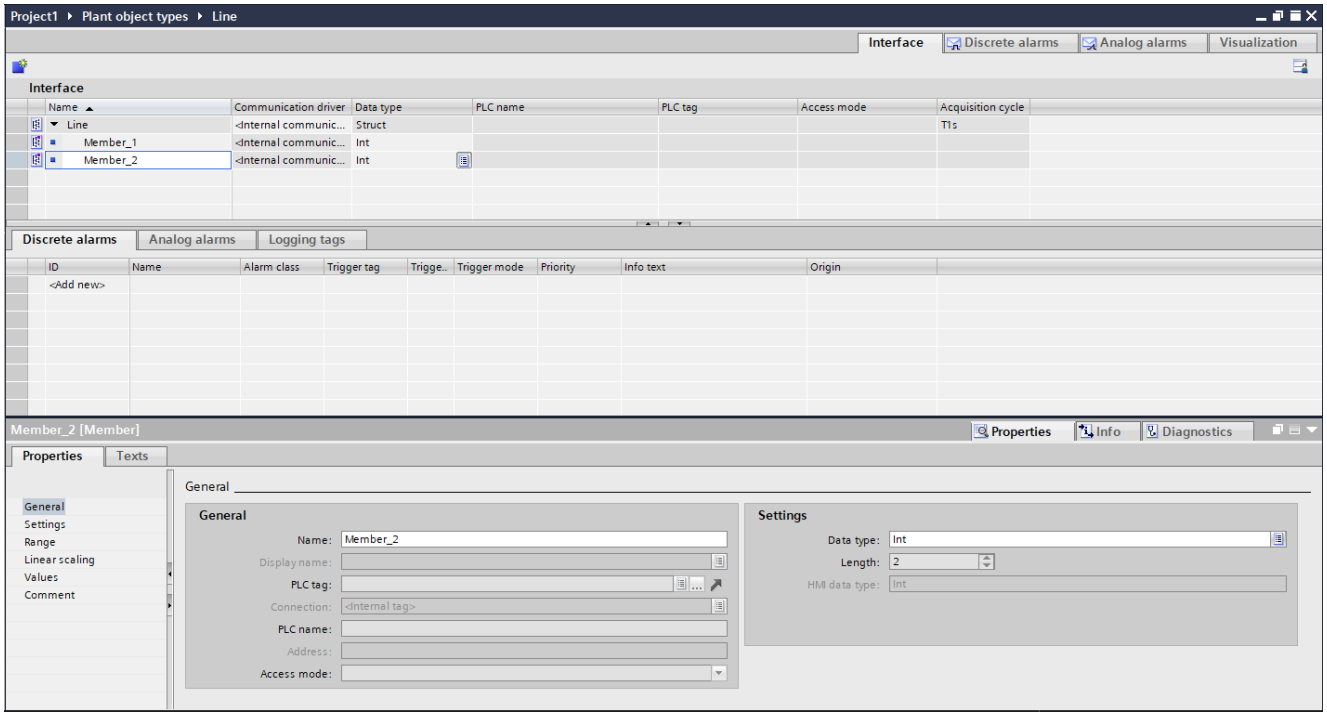
Configuration without using function blocks is described below.

Requirement

- The WinCC Unified PC RT HMI device has been created.
- A plant view has been created and assigned to the HMI device.
- A plant object type has been created.
- A SIMATIC S7-1500 PLC has been created.
- Tags have been configured in the S7-1500.

Procedure

1. Double-click the plant object type in the "Plant object types" editor.
   An empty plant object type with the "Struct" data type appears under "Interface".
2. To add data members to the plant object type, select the plant object type and click "Insert object".



The created data member inherits all properties from the higher level plant object type. "Internal communication" is selected by default in the column "Communications driver" for the newly created data members of the plant object types.

3. If you want to configure an external data member, select "SIMATIC S7-1500" in the "Communication driver" column.
4. Assign a PLC tag to the external data member in the "Tag" column.

Note

If an HMI device is assigned to the plant view, it is possible to view the data members in the "HMI tags" editor in the "Plant object tags" tab. You also have write rights for the "Comment" column.

You can also use the configured data members in screens of an assigned HMI device, e.g., for dynamization instead of tags.

Tips for an efficient procedure

- Differentiate between identically named plant objects and plant object types using the "Insert object" button in the "Interface" tab. If the "Insert object" button is enabled, you have selected a plant object type. If you have selected a plant object, the "Insert object" button is disabled.

See also

Creating plant object types (Page 7071)

Structure of a plant model (Page 7061)

Creating plant objects (Page 7072)

Assigning a plant hierarchy to a HMI device (Page 7070)

Configuration concept (Page 7058)

Creating a plant hierarchy (Page 7069)

Configuring plant object types from the data blocks of an S7-1500 (Page 7075)

Assigning process data to plant objects (Page 7076)

15.3.2.4 Configuring plant object types from the data blocks of an S7-1500

Introduction

Configure the plant object types either from the data blocks of an S7-1500 or define the properties and the external and internal data members for the plant object types without connection to a PLC.

In both cases you can extend the structure the created plant object types with additional internal or external data members.

Configuration from the data blocks of an S7-1500 PLC is described below.

Configure plant object types from the configured program blocks of an S7-1500 PLC using drag-and-drop operation.

Requirement

- The WinCC Unified PC RT HMI device has been created.
- A plant view has been created and assigned to the WinCC Unified PC RT.
- A plant object type has been created.
- A SIMATIC S7-1500 PLC has been created.
- A function block is configured in the SIMATIC S7-1500 PLC.

Procedure

1. Open the plant object type in the "Interface" tab.
2. Set the "Communication driver" parameter to the SIMATIC S7-1500 PLC that contains a configured function block.
3. Navigate to the function block in the PLC.
4. Select the function block.

5. Drag the function block to the "PLC tags" field in the "Interface" tab.
   The corresponding structure with data members based on the function block is created in the "Interface" tab.
   When you edit the blocks of the PLC, these changes are automatically transferred to the plant object types.
6. To add additional data members to the plant object type, select the plant object type and click "Insert object".

   Note

   You can create additional data member for each plant object type.

   Function blocks (FBs) or PLC user data types act as basis for the configuration of the plant object types and their data members.

   A member structure can also be connected to a PLC type, for example, function block (FB) or PLC user data type.

   "Raw" data types and arrays are also supported.

   An assignment of the controller blocks to the external data members of the plant object types is only possible if names and data types are identical in the PLC and in the plant object type.
7. If necessary, adjust the data type of the data member.
8. To clear the connection between of a controller and the data members object, delete the block in the "PLC tag" column or select "None".

Tips for an efficient procedure

- Differentiate between identically named plant objects and plant object types using the "Insert object" button in the "Interface" tab. If the "Insert object" button is enabled, you have selected a plant object type. If you have selected a plant object, the "Insert object" button is disabled.

Note

If an HMI device is assigned to the plant view, it is possible to view the data members in the "HMI tags" editor in the "Plant object tags" tab. You also have write rights for the "Comment" column.

You can also use the configured data members in screens of an assigned HMI device, e.g., for dynamization instead of tags.

See also

Configure plant object types (Page 7073)

Assigning process data to plant objects (Page 7076)

15.3.2.5 Assigning process data to plant objects

Introduction

To establish the communication between a S7-1500 controller and a WinCC Unified PC RT device, connect a plant object with a PLC tag or a data block of the PLC.

Requirement

- An S7-1500 PLC and a WinCC Unified PC RT HMI device are configured and connected.
- At least one plant object type in the project contains PLC user data types or function blocks (FBs).

Procedure

1. Drag a plant object type to the plant view.
   The plant object is created based on the plant object type.
2. Double-click the plant object.
3. In the "Interface" tab, in the "Connection" column, select the configured HMI connection for all external data members of the plant object type.
   Select only between the HMI connections that are created for the S7-1500 controllers available in the project.
4. In the "PLC tag" column, select a PLC tag.

Note

In the "Interface" tab, similar to in the "HMI tags" editor, you can view or edit the properties in the following areas:

- "General"
- "Settings"
- "Range"
- "Linear scaling"
- "Values"
- "Comment"

See also

Configure plant object types (Page 7073)

Configuring plant object types from the data blocks of an S7-1500 (Page 7075)

15.3.3 Configuring screens

15.3.3.1 Basic information on configuring screens

Overview

The configuration of screens for operating and monitoring is also available to you in object-oriented configuration. This means that you are working in two areas, under "Project tree> Devices" and "Project tree > Plant objects > Visualization". Here you work with both screens and faceplates that also support the type-instance concept.

In the area "Project tree > Devices", configure screens for HMI devices as usual. In the screens, also configure companion controls that are relevant for the display of screens of the plant objects.

In the "Project tree > Plant objects > Visualization" area, you configure screens for plant objects.

In the "Plant object types > Visualization" area, configure faceplates for plant object types.

In the areas under "Project tree > Devices" and "Project tree > Plant Objects > Visualization", the same predefined screen objects are available in the "Toolbox" task card.

When configuring faceplates, a minimized tool area is available under "Toolbox".

Configuration options

Under "Project tree > Devices", you configure a screen for the created HMI device with the "Plant overview" control and one of the companion controls, such as a screen window. In runtime, navigate the plant structure to the plant objects via the "Plant overview" control. The screen windows in the plant overview display the screens that you have previously configured for the plant object.

The companion controls are connected to one another and supplement one another in displaying the data values.

The following controls can act as companion control for the plant overview:

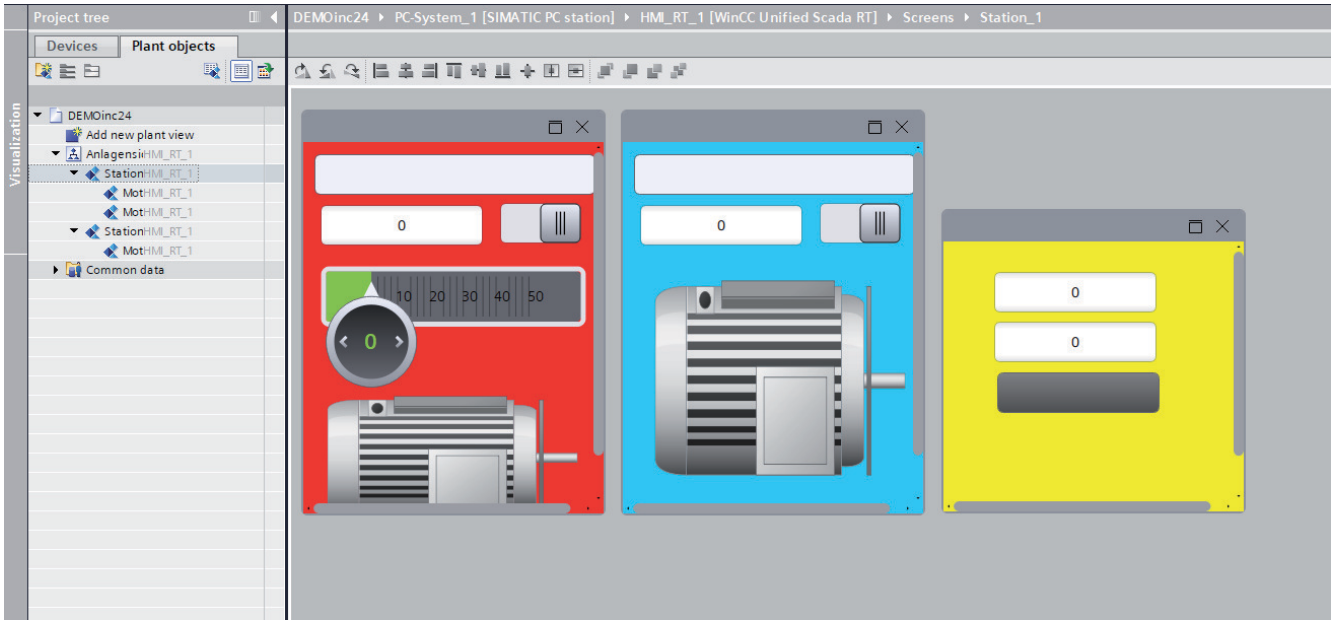
- Alarm control
- Screen window
- Calendar control (when using the WinCC Unified Calendar option)

When required, configure controls as usual, e.g. screen windows, alarm control, or trend control. Select the specific plant object in the plant hierarchy as data source for the alarm control and trend control. Configure the alarm control and trend control for plant objects on the basis of the data members of the plant object types. The procedure for configuring these controls does not differ from the procedure for the device-specific configuration.

The following options are available in runtime:

- Display the hierarchy path of the alarm source
- Display the hierarchy path of the trend values
- Filter the alarm control by plant objects
- Display process values for the selected plant object

Use the "Visualization" tab for the direct visualization of plant objects and plant object types. You can create one screen for each plant object, and you can link one faceplate type for each plant object type. The type-instance concept is used for configuring the screens for plant objects and plant object types. All relevant elements are contained in the faceplate type of a plant object type. Drag a plant object into the screen. A faceplate container is created. For example, several faceplates can be integrated into one visualization of a higher-level plant object.



Configuration steps

In general, proceed in the following order when configuring the screens for your plant:

1. Configure plant object types.
2. Configure faceplates for plant object types.
3. Create plant objects from plant object types.
4. Create screens for plant objects.
5. For the display in runtime, configure the "Plant hierarchy" and "Screen window" controls in a screen of the HMI device.

Displaying plant objects and plant object types

In general, you do not have to create a screen for each plant object. Create an overview for the higher-level plant object. Then create faceplates for the plant object types and drag them to the overview screen of the higher-level plant object using drag-and-drop.

If the screens of the plant objects need to differ from the screens used for the plant object types, you can use the faceplates of the plant object types as a basis and configure additional elements.

In runtime, select a plant object in the "Plant overview" control to display its screen.

See also

Configuring screens for plant objects (Page 7080)

Example: Configuring screens for brewery production lines (Page 7103)

Configuring faceplates (Page 524)

15.3.3.2 Configuring screens for plant objects

Introduction

You configure an overview screen for the higher-level plant object with multiple faceplate containers for lower-level plant objects, for example, for a station that has lower-level objects motor and conveyor belt.

For each plant object you can configure a screen in which all lower-level plant objects are visible. To do this, use the faceplate containers of the plant object types.

If necessary, you also configure basic objects, elements and controls in the screen. For example, you use I/O fields to display process values such as status, temperature and rate.

In the following, you will obtain the data to be processed, such as temperature measurements or speed values from the data blocks of a controller.

You represent lower-level plant objects using faceplates in the overview screen of the higher-level plant hierarchy.

In runtime, the screen window technology assists you in switching between plant objects and representing multiple plant objects in a screen.

You can also use the "Plant overview" control to set up screen navigation via the plant. In runtime, you monitor the plant in this manner and see the overall progress at a glance.

Requirement

- A SIMATIC S7-1500 has been configured in the project.
- WinCC Unified Runtime is configured in the project.
- A plant object type has been created.
- The plant view has been created with the plant objects and assigned to WinCC Unified PC RT.
- The interface tags of the plant object types are linked to the S7-1500.

Procedure

1. Open the plant object editor.
2. In the "Visualization" tab, click "Add new screen".
   A screen is created.
3. If necessary, edit the width and height of the screen under "Properties" in the inspector window.
4. If necessary, configure the required elements and controls for the plant object, such as I/O fields and text fields.

5. Change to the "Libraries" task card.
6. Open the project library.
7. Create a faceplate type.
8. Configure the required screen objects, interface tags and interface properties in the faceplate type and release the version.
9. Open the editor of the plant object type.
10. In the "Visualization" tab, drag the faceplate type from the project library to the "Drop faceplates here" button.
11. Connect the faceplate tags to the interface tags of the plant object type.
12. Create a plant object from the plant object type using a drag-and-drop operation.
13. Open the plant object editor.
14. Assign the plant object a PLC tag under "Interface" in the "PLC tag" column:
15. To display the faceplate container of the plant object type in the screen, drag the plant object from the plant view to the configured screen.

Tips for an efficient procedure

- Adjust the position of the faceplate container in the overview screen using the mouse or the corresponding icons on the toolbar.
- You can zoom in and out of the faceplate container in the overview screen.
- You can at any time delete and reconfigure the overview screen which contains the faceplate container for lower-level plant objects. You can reuse the faceplate types at any time.

Note

If the screen area is not sufficient for all faceplate containers, the faceplate containers are superimposed on each other in runtime.

Note

If additional basic objects, elements and controls are required specifically for a plant object, you can use the faceplates of the plant object types as a basis and configure additional objects.

Note

Note that interface mapping is deleted during updates of faceplate types used in plant objects.

Result

You have created a screen with a faceplate instance for the plant project.

See also

Operating "Plant overview" in runtime (Page 7114)

Basic information on configuring screens (Page 7077)

Example: Configuring screens for brewery production lines (Page 7103)

Configuring an alarm control for plant objects (Page 7084)

15.3.4 Configuring the controls

15.3.4.1 Configuring "Plant overview" control and companion controls

Introduction

You require the control "Plant overview" when you want to navigate through the plant.

The companion controls are connected to one another and supplement one another in displaying the data values.

You require the companion controls for the following displays:

- Display plant object screens and screen windows using the navigation option throughout the entire plant (plant overview and screen windows)
- Display alarms for plant objects using the navigation option throughout the entire plant (plant overview and alarm view)

The following controls can act as companion control for the plant overview:

- Alarm control
- Screen window
- Calendar view (when using the WinCC Unified Calendaroption)

Requirement

- A screen is open.
- The "Toolbox" task card is open.

Procedure

1. Insert the "Plant overview" control from the "Toolbox > My controls" task card into the screen.

2. Add a companion control.
   Select from the following controls:
   - Alarm control
   - Screen window
   - Calendar control

   Note

   As companion controls, you can only select controls already configured in the screen.

3. Select the "Plant overview" control.
4. Open the Inspector window under "Properties > Properties > Miscellaneous > Interface > Companion control".
   The "Companion control" editor opens on the right-hand side of the Inspector window.
5. Click "Add".
   An element (starting with 0) is created.
6. Create an element for each companion control:
7. Specify the control type for each element:
   - Alarm control
   - Display window (for screen windows)
   - Calendar control (for calendar control)
8. Define the respective companion control as control reference for each element.

9. Specify a filter:
   - No filters: You see all plant objects
   - By plant objects for which alarms are available
   - By plant objects for which screen windows are configured
10. Specify the navigation type:
    - Static: The plant tree is displayed completely in runtime.
    - Dynamic: In runtime, you specify as of which level the plant tree is displayed by double-clicking a plant object.
      The levels below the selected plant object are available and are expanded.
      You can always navigate to the next higher level in runtime.

    The buttons of the toolbar and the filter bar relate to the displayed area.
11. Specify the root node. You have the following options:
    - In the "Static value" column, specify the path of the plant object according to the following schema "HMI_device.hierarchy::Plant view/Plant object", for example, "HMI_RT_1.hierarchy::Brewery/Bottling".
    - Specify the root node dynamically using a tag or a script.

    If a root node is configured, the root node and all objects below the root node are available in the plant overview.
12. Specifies whether the toolbar is shown.
13. Specifies whether the menu bar is shown.

Result

In runtime you see the screen with the "Plant overview" control and the companion controls. When you navigate to the respective plant object in the "Plant overview" control, the content of this plant object is displayed in the companion controls.

If you have configured the screen window as companion control, navigate in runtime in the "Plant overview" control through the plant and display the screens you have configured for the respective plant object.

If you have configured the alarm control as companion control, navigate in runtime in the "Plant overview" control through the plant and have the alarms for the plant objects displayed in the alarm view.

15.3.4.2 Configuring an alarm control for plant objects

Overview

Configure an alarm control, as in the device-specific configuration in a HMI device screen. In order that the alarm control can display the alarms of the plant objects, assign the plant hierarchy to your HMI device.

To directly jump to the alarms of the plant objects in runtime, configure the alarm control is companion control to the "Plant overview" control.

To filter by plant object alarms in the alarm control, configure a filter with the criterion "Area" with one of the following two conditions:

- "Equal" - only shows the alarms of the selected plant object in runtime.
- "Begins with" - shows the alarms of the underlying objects of the selected plant object in runtime.

Configuring a filter for plant objects

To filter by plant object alarms in the alarm control, configure a filter as follows:

1. In the Inspector window under "Properties > Filter", click in the "Static value" column. The "Alarm filter configuration" dialog opens.
2. Select the "Area" criteria.
3. Select the condition "Equal"
4. Click on the selection list in the "Operand" column.
5. Select the plant object whose alarms you want to display in runtime.

Note

You can also create filter criteria directly in runtime and use them as filters.

Result

Alarms for the selected plant object are displayed in runtime.

See also

Configuring screens for plant objects (Page 7080)

Configuring an alarm control (Page 753)

Displaying alarms for plant objects in runtime (Page 7120)

15.3.4.3 Configuring trend control for plant objects

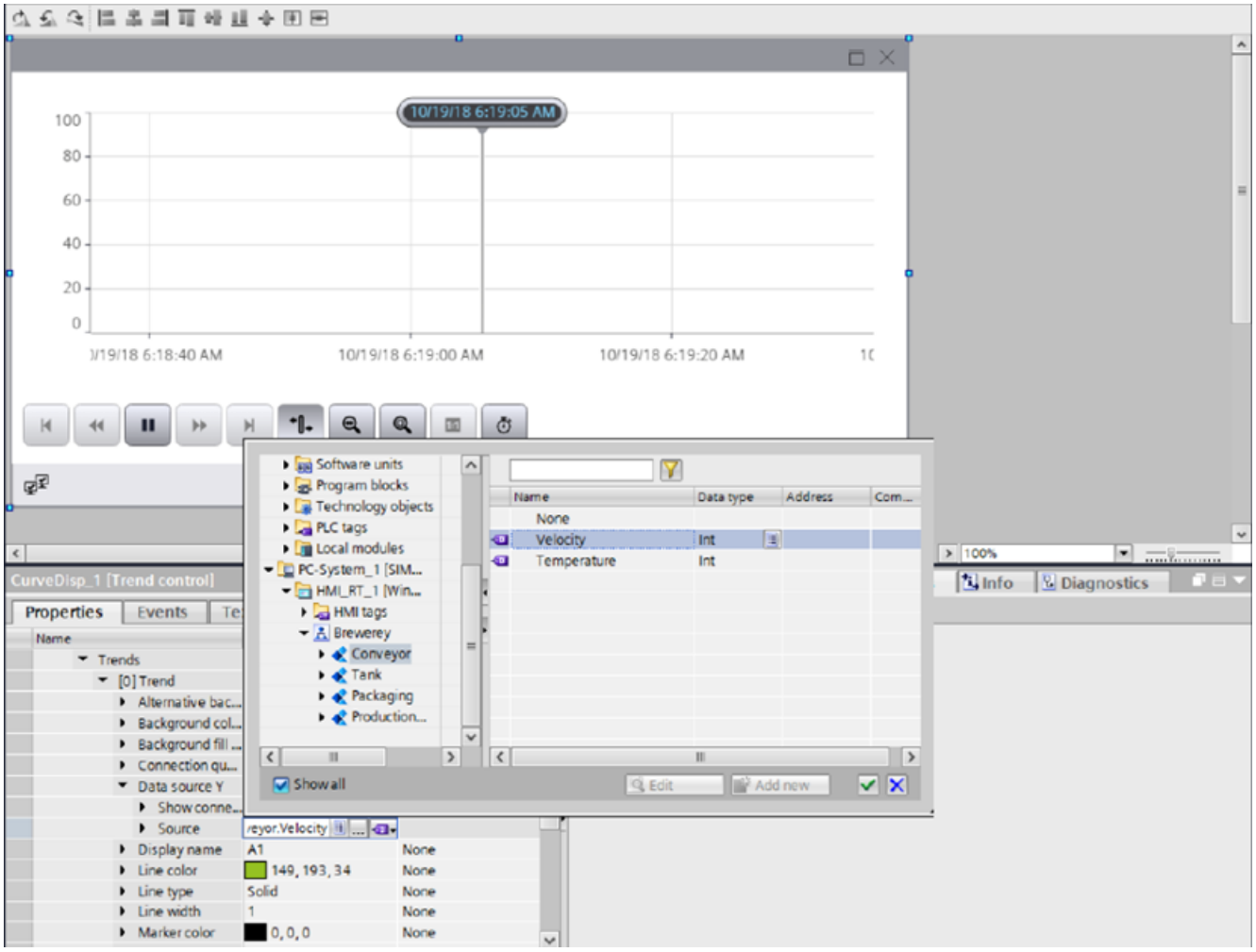
Overview

A trend control, as in the device specific configuration in a HMI device screen. Assign the plant view to your HMI device in order that the trend control can graphically represent the values of the data members of the individual plant objects in runtime.

The trend control allows you to display current and logged values for a specific time window, for example.

As with device-specific configuration, when you configure the trend control for the display of the data values you define the sources from which the values are obtained on the HMI device in runtime. The following sources are available:

- Current process values from data members of the plant object types
- Archived values from logging tags



The path of the plant object is shown in the trend control when displayed in runtime.

See also

Configuring the logging of plant object types (Page 7094)

Configuring a trend control (Page 680)

Display process data of the plant objects in a trend control (Page 7116)

15.3.5 Configuring alarms

15.3.5.1 Basic information on configuring alarms

Overview

In object-oriented configuration, as in device-specific configuration via the alarming, events from the monitoring function in WinCC are displayed in form of alarms. The alarms can be acknowledged by the operator and, if necessary, logged. To do this, configure alarms that are separated into alarm classes.

For plant objects you can configure the following alarms that are used to monitor the plant:

- Discrete alarms: Display status changes
- Analog alarms: Display limit value violations (value changes),

Configure bit or analog alarms for plant object types on the basis of internal or external data members. From these plant object types you create plant objects.

If you have configured an alarm system for plant objects you can display the hierarchy path of the alarm source and the alarm status of a line or a machine in runtime, filter the alarm control by plant objects and navigate to the alarm source.

You can also filter the most frequently occurring alarms by plant object and only permit the alarm of the respective object and all lower-level objects to be displayed.

Configuration steps

In general, proceed in the following order when configuring the alarms for the plant objects:

- Configure plant object types
- Configuring bit or analog alarms for plant object types on the basis of data members.
- Creating plant objects from plant object types
- Configuring alarm control in a screen
- Configuring "Plant view" control as companion control for the alarm control.

Note

An alarm is linked to the respective plant object type. If you delete the plant object type the alarm will also be deleted.

Configuring an alarm view

The alarm view is configured for a screen. Current or logged alarms are displayed in the alarm view in runtime. More than one alarm can be displayed simultaneously, depending on the configured size. Configure the criteria for alarm filtering.

You can also configure multiple alarm views with different contents and in different screens.

See also

Configure discrete alarms for plant objects (Page 7088)

Configuring analog alarms for plant objects (Page 7091)

15.3.5.2 Configure discrete alarms for plant objects

Introduction

If you have configured an alarm system for plant objects you can display the hierarchy path of the alarm source and the alarm status of a line or a machine in runtime, filter the alarm control by plant objects and navigate to the alarm source.

You can also filter the most frequently occurring alarms by plant object and only permit the alarm of the respective object and all lower-level objects to be displayed.

An alarm is linked to the respective plant object type. If you delete the plant object type the alarm will also be deleted.

Requirement

- A plant object type with associated external or internal data members (with elementary data types) has been created.
- The plant structure has been assigned to a device.

Procedure

1. Select the respective data member of the plant object type on the basis of which you want to configure an alarm.
2. To create a new discrete alarm, double-click on "<Add>" under "Discrete alarms" in the table. A new discrete alarm is created.
3. Assign a name for the alarm.

   Note

   The name of a discrete alarm can contain up to 128 characters.

4. To configure the alarm, select "Properties > General" in the Inspector window:
   - Enter the alarm text.
   - Change the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").

   Note

   The alarm text must be unique in the context of the plant object type. Hierarchical information is not permitted in the alarm text.

   Note

   You can use the priority to sort or filter the alarms in the alarm control. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display area in a single-line alarm control.

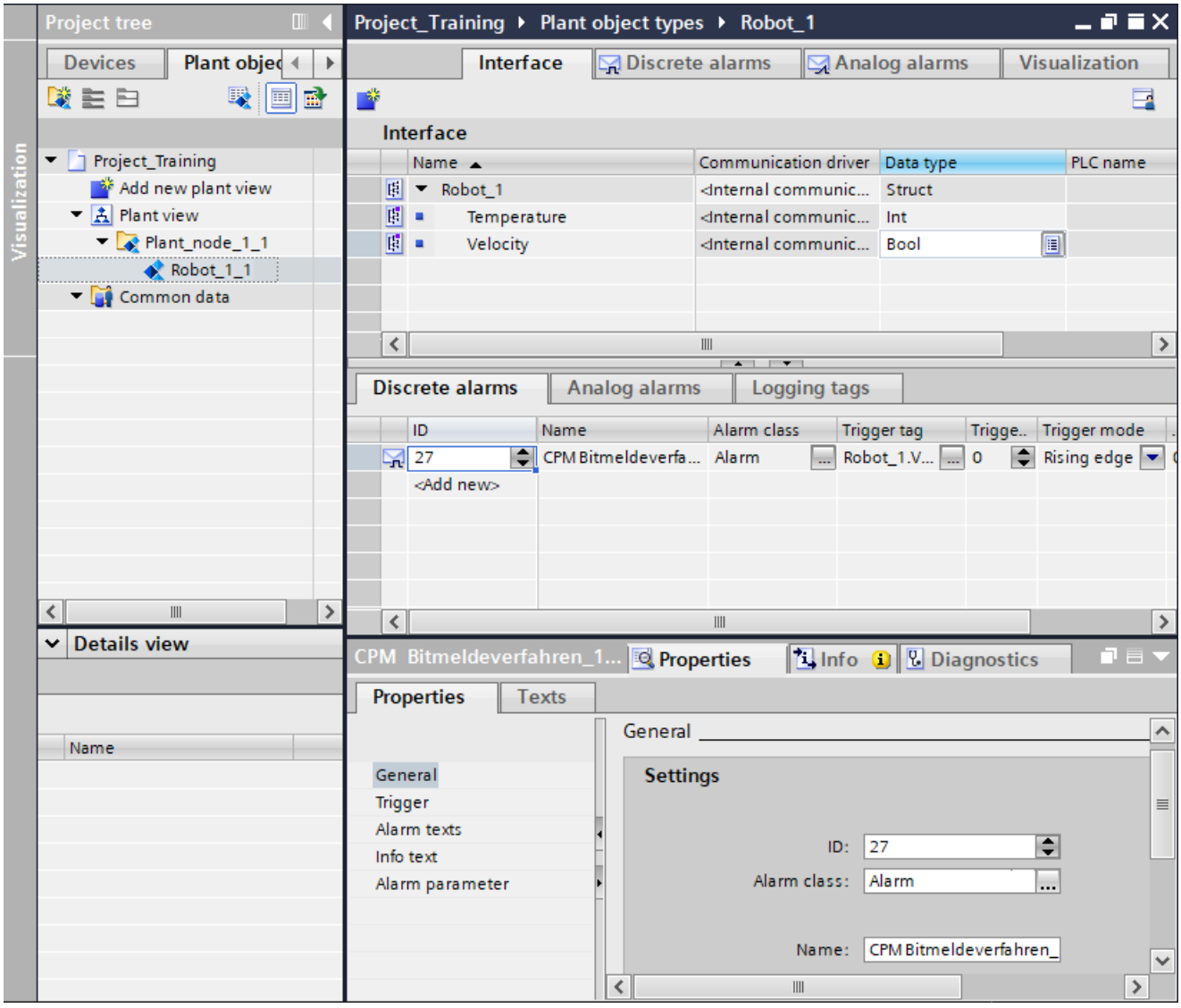
   If you filter the alarm control by priority "16", only the alarms with priority "16" will appear.

5. Select "Properties > Trigger" in the Inspector window to select the tag and the bit that triggers the alarm.

   Note

   Only the data member of the plant object type is permitted as trigger tag.

6. Select "Trigger mode" to specify whether to trigger the alarm at a rising or falling edge.
7. To configure the alarm text, select "Properties > General > Alarm text".
   – Enter the text for the alarm under "Alarm text".



See also

Configuring analog alarms for plant objects (Page 7091)

Basic information on configuring alarms (Page 7087)

Displaying alarms for plant objects in runtime (Page 7120)

15.3.5.3 Configuring analog alarms for plant objects

Introduction

If you have configured an alarm system for plant objects you can display the hierarchy path of the alarm source and the alarm status of a line or a machine in runtime, filter the alarm control by plant objects and navigate to the alarm source.

You can also filter the most frequently occurring alarms by plant object and only permit the alarm of the respective object and all lower-level objects to be displayed.

An alarm is linked to the respective plant object type. If you delete the plant object type the alarm will also be deleted.

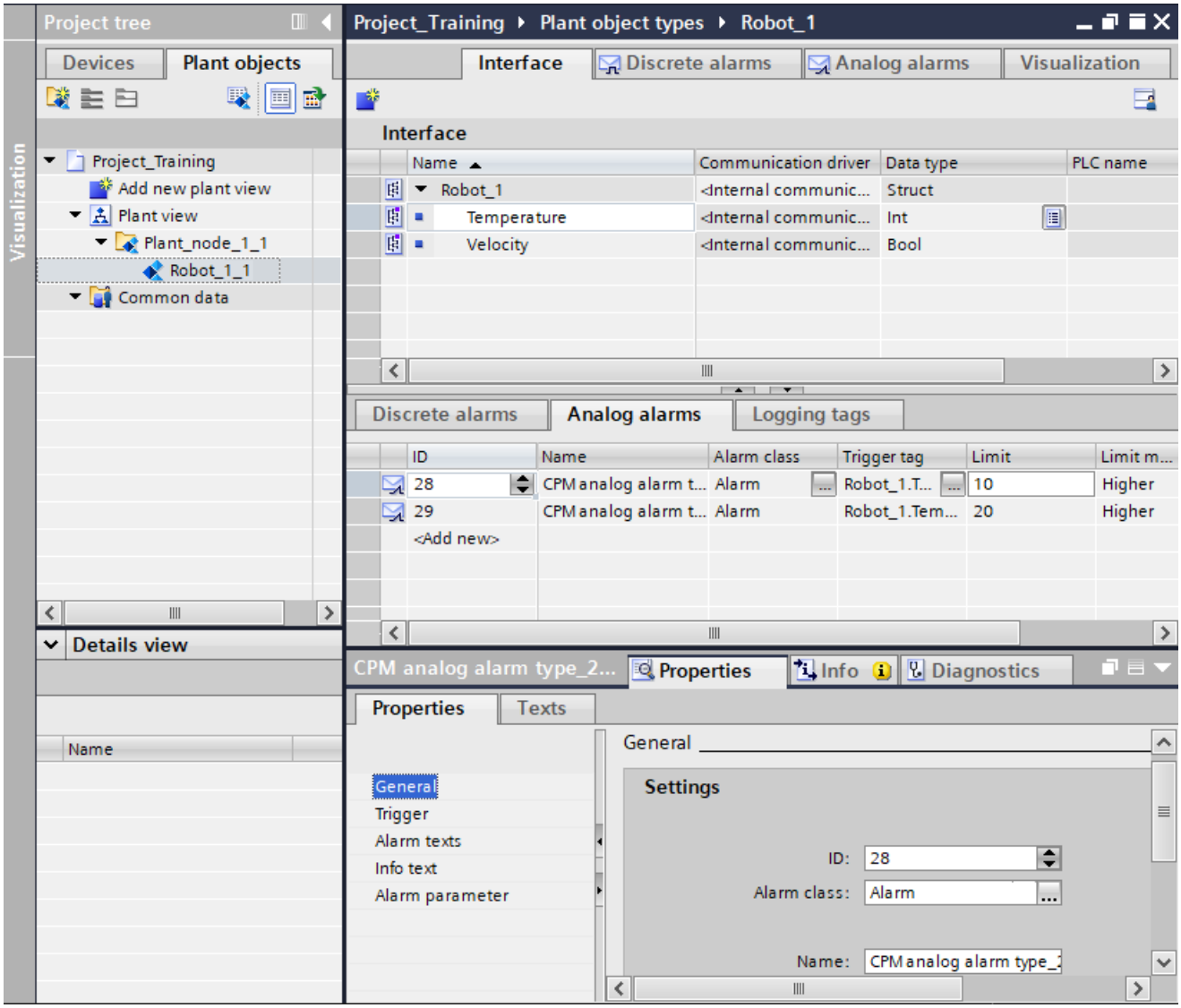
Requirement

- A plant object type with associated external or internal data members (with elementary data types) has been created.
- The plant structure has been assigned to a device.

Procedure

1. Select the respective data member of the plant object type on the basis of which you want to configure an alarm.
2. Enter the alarm text under "Properties > General".
3. To create a new analog alarm, double-click in the table on "<Add>" under "Analog alarms" in the table.
   A new alarm is displayed.

4. To configure the alarm, select "Properties > General" in the Inspector window:
   - Enter the alarm text.
   - Change the name of the alarm as required.
   - Select the alarm class.
   - Configure the priority of the alarm (a value of between "0" and "16").



Note

The alarm text must be unique in the context of the plant object type. Hierarchical information is not permitted in the alarm text.

Note

You can use the priority to sort or filter the alarms in the alarm control. With sorting by priority, you can ensure that the most important alarm (high priority) is shown in the display area in a single-line alarm control.

If you filter the alarm control by priority "16", only the alarms with priority "16" will appear.

5. In the Inspector window, select the tag that triggers the alarm, e.g. a data member, under "Properties > Trigger".

Note

Only the data member of the plant object type is permitted as trigger tag.

6. In the Inspector window under "Properties > Trigger", enter a limit in the "Value" field in the "Limits" area.
7. Select the trigger mode in the "Mode" field:
   - "Lower": The alarm is triggered if the limit is undershot.
   - "Upper": The alarm is triggered if the limit is exceeded.
   - "Equal": The alarm is triggered when the limit is reached.
   - "Not equal": The alarm is triggered if the limit is not reached.
   - "Lower or equal": The alarm is triggered if the limit is undershot or reached.
   - "Greater or equal": The alarm is triggered if the limit is exceeded or reached.
8. You can create additional limits for the alarm, if necessary. Note the following:
   - A tag is monitored using only one alarm type. You should therefore create either analog alarms **or** discrete alarms for a tag.
   - If the object included in the selection does not yet exist, create it in the object list and change its properties later.
9. Select the analog alarm to which you want to assign the limits.

See also

Configure discrete alarms for plant objects (Page 7088)

Example: Configuring analog alarms for temperature monitoring (Page 7106)

Basic information on configuring alarms (Page 7087)

Displaying alarms for plant objects in runtime (Page 7120)

15.3.6 Configuring the logging of plant object types

Introduction

Save the values of the data members of the plant object types in logs for later evaluation. Alarm logging can be used to analyze error states, to optimize maintenance cycles, and to document the process.

Create a logging tag for each data member of the plant object type. These logging tags are saved in the data log of the assigned device.

You can analyze the logged tag values directly in your project, such as in a trend view, or in another user program, such as Excel.

The logging tags are created for the plant object types. This means that the plant objects are automatically supplied with the logging tags of the plant object types.

Requirement

- The plant hierarchy has been created and assigned to a device.
- A plant object type with associated external or internal data members (with elementary data types) has been created.

Procedure

1. Under "Interface", jump to the "Logging tags" tab in the middle part of the work area.
2. Under "Interface", select a data member of a plant object type that you want to log.
3. Click "Add" under "Logging tags".
   A logging tag is created.
   The logging tag is linked to the tag. The data type of the logging tag corresponds to the data type of the connected tag.

   Note

   A logging tag is automatically assigned to a data log. This assignment cannot be changed.

   The assignment is only possible if the plant hierarchy is assigned to a HMI device.

4. Specify the logging mode.

   Note

   Depending on the configuration, the database can grow very quickly. This can occur, for example, when you select a short cycle without smoothing and without compression.

5. When the "Cyclic" logging mode is set, define the logging mode and the factor under "Properties > Properties > Cycle".
6. Define the tag trigger depending on the logging mode.

7. Define the limit values.

   Note

   Process values that are outside the set limit range will not be logged.

8. Define the smoothing.
9. If you have selected the "Cyclic" logging mode, define the compression.

Logging tags and logging is available under "Visualizing processes with Runtime Unified".

See also

Configuring trend control for plant objects (Page 7085)

15.3.7 Good Manufacturing Practice

Traceability and therefore the documentation of production data is becoming increasingly important in many sectors such as the pharmaceutical industry, the food and beverage industry, and the related mechanical engineering industry.

Therefore, sector-specific and cross-industry standards have been developed for the electronic documentation of production data.

The most important set of regulations is the FDA guideline 21 CFR Part 11 for electronic data records and electronic signatures issued by the FDA, the US Food and Drug Administration. In addition, different EU regulations apply, such as EU 178/2002, depending on the industry.

Requirements for production systems in these industries have been developed on the basis of 21 CFR Part 11 and the corresponding layout to comply with GMP (Good Manufacturing Practice). They are also required for other industries.

The following main requirements are derived from these directives and regulations:

- Creation of an Audit Trail or operating trace in runtime
  Based on this document, it is possible to trace the user who carried out the operator action on the machine at what time.
- Important process steps must also be assigned to a clear responsibility, for example, via an electronic signature.

GMP (Good Manufacturing Practice)

If necessary, activate the GMP-compliant configuration in the Runtime settings.

GMP is also displayed in the properties of a tag of a PLC user data type used in an HMI device even if it is not enabled in the specific device but the tag is used in an additional device that uses GMP. The menu command "GMP" is grayed out in this case and cannot be edited.

15.3.8 Example

15.3.8.1 Example: Scenario

The production lines "Bottling" and "Packaging" exist alongside other lines in a brewery and are connected to one another in the production chain. The production lines consist of multiple units.

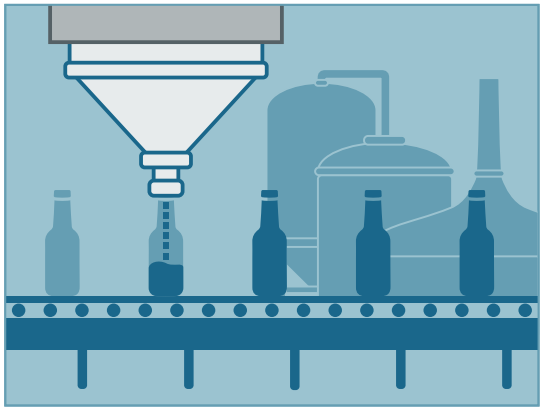
You configure plant objects in the plant view from the plant object types based on the data blocks.

Screens must be configured for all plant objects relevant for the monitoring. In addition, a screen navigation should be set up so that the user can navigate from one plant object screen to another in runtime using the "Plant overview" control. The production value should also be monitored, for example the temperature in the filling tank or the weight of the product after the filling. Make sure to notify the operator in case of deviations.

The relevant production values must be logged for quality assurance purposes and for Food Authority audits.

The "Bottling" production line consists of the following units:

- Conveyor belt 1 ("Conveyor_1")
- Robot 1 ("Robot_1"): Places the bottles on the conveyor.
- Filling tank ("Filling Tank"): Fills the bottles, the temperature in the filling tank is monitored.
- Robot 2 ("Robot_2"): Closes the bottles.
- Robot 3 ("Robot_3"): Performs quality checks (weight and light barrier)

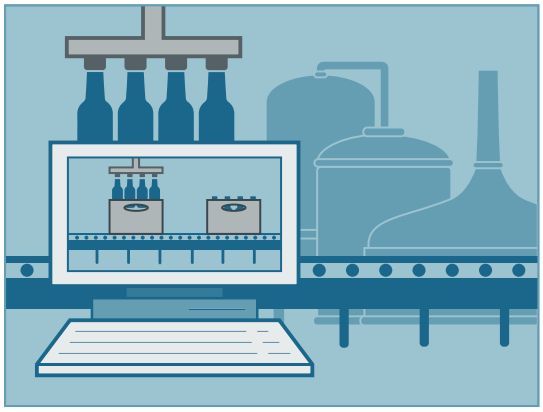


The bottles are sorted from the conveyor belt into beverage crates on the "Packaging" production line.

The "Packaging" production line consists of the following units:

- Conveyor belt 2 ("Conveyor_2"): Makes the filled bottles available.
- Conveyor belt 3 ("Conveyor_3"): Conveys filled crates.
- Robot 4 ("Robot_4"): Places the crates on the conveyor belt.
- Robot 5 ("Robot_5"): Places the bottles in the crates.

- Robot 6 ("Robot_6"): Places the crates on the pallet.
- Robot 7 ("Robot_7"): Performs quality checks (weight and light barrier)



15.3.8.2 Example: Implementation concept

Creating a plant view

They map the plants, units and production lines of the brewery in the plant view. To do this, you first create the plant object types that you can reuse for plant objects. Based on the plant object types, you then create the "Brewery" plant view from the plant objects. The two production lines "Bottling" and "Packaging" contain the units according to the following concept:

- Bottling
  - Robot_1
  - Robot_2
  - Robot_3
  - Conveyor_1
  - Filling Tank_1
- Packaging
  - Robot_4
  - Robot_5
  - Robot_6
  - Robot_7
  - Conveyor_2
  - Conveyor_3

Visualization

You create faceplate types for the "Conveyor", "Robot" and "Filling Tank" plant object types.

The faceplate types of the plant object types are instantiated in the screens of the plant objects.

According to the structure of the production line, the "Robot" faceplate type is reused three times in the "Bottling" production line and four times in the "Packaging" production line.

You also configure an overview screen called "Overview" with the "Plant overview" control. The plant overview gives you direct access to the unit data in runtime.

Temperature monitoring

You want to monitor the temperature in the filling container and to notify the operator if changes occur.

To monitor the temperature, configure a trend control and an alarm control for the filling tank as companion controls for the plant overview.

Because a specific temperature must not be exceeded during the bottling of beverages, configure analog alarms for the filling tank that can be output via the alarm view.

The trend view provides you with an overview of the temperature trend of the filling tank.

Logging

The production values are logged for the Food Authority inspections. It must be verified that the temperature was complied with and that quality checks were regularly performed. To this end, configure logging tags for the relevant plant objects.

15.3.8.3 Example: Determine plant object type

Scenario

For a new brewery location, two employees of an engineering office configure the process visualization and plant-specific parameters. The employees develop a configuration concept for this.

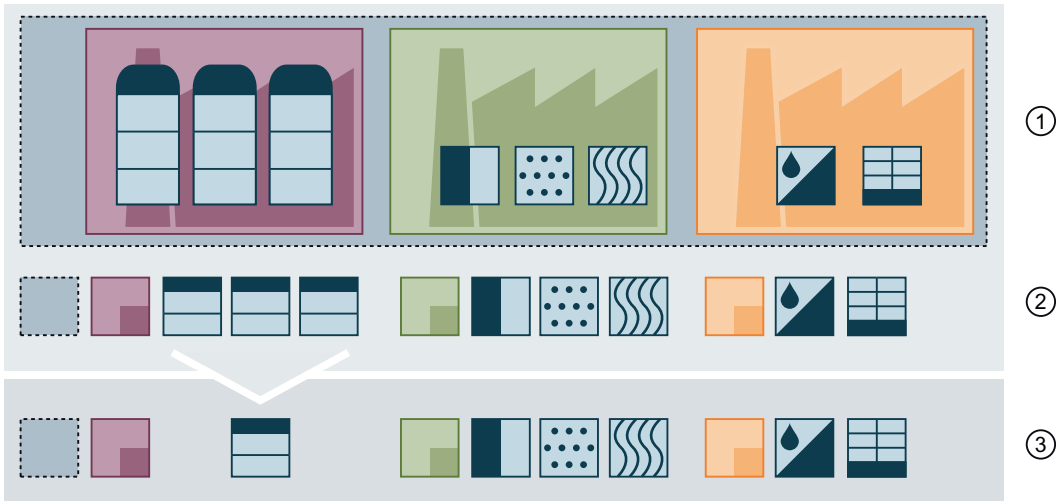
The following examples shows how process visualization and object-oriented configured mesh with each other.

Determining plant object types

How you determine plant object types by analyzing the plant structure depends on the context:

- In the WinCC Runtime Unified context, you view the plant "from the bottom" in the process view. Functional units are in the background compared to plant objects of the field and process levels. The functional units are still necessary for a complete mapping of the plant.
- In the context of plant-specific KPIs, look at the plant "from the top" in the plant view. Plant objects of the field and process levels may no longer be relevant.

The following figure is a schematic representation of the analysis sequence of the plant structure for defining the plant object types:



① Analyze the plant structure: The brewery consists of the three plant units, "Delivery/Storage", "Processing" and "Bottling". Various processes run in the plant sections.

② Determining relevant plant objects for monitoring process and productivity: These plant objects are the basis for mapping the plant hierarchy.

③ Defining plant object types: Plant objects used multiple times are mapped using a common plant object type. The data structure and context information is configured for each plant object type.

Brewery

- Plant-specific parameters: Cumulative characteristics of productivity

Unit for delivery and storage of ingredients

- Plant-specific parameters: Characteristics for duration of delivery

Tank for storage of ingredients

- Process visualization: Monitoring temperature and fill level

Unit for processing

- Plant-specific parameters: Characteristics of productivity

"Mashing"

- "Batches taken out of mash" Process visualization: Monitoring temperature and pressure
- "Pump back batches taken out of mash", check: "Iodine test": Process visualization: Monitoring temperature and pressure

"Purifying" Process visualization: Monitoring temperature and pressure

"Wort boiling" Process visualization: Monitoring temperature and pressure

- Whirlpooling
- Addition of yeast and fermenting
- Storage

Unit for bottling and packaging

- Plant-specific parameters: Characteristics for period of change in production, unplanned downtime and produced quantities

Bottling process

- Process visualization: Monitoring temperature / control of filling

Process for packaging the beer bottles on pallets

- Process visualization: Monitoring packaging

Note

The data structure of a plant object type is often reflected in the function blocks of the user program. In such a case you can create plant object types automatically. Consultation with the programmer is recommended especially for plant object types with strong links to the PLC.

See also

Creating plant object types (Page 7071)

15.3.8.4 Example: Creating a plant view

Task

You map the brewery plant with its units and production lines with the plant view and make the unit data available for the entire project.

You create the "Brewery" plant view and assign it to the HMI device.

Requirement

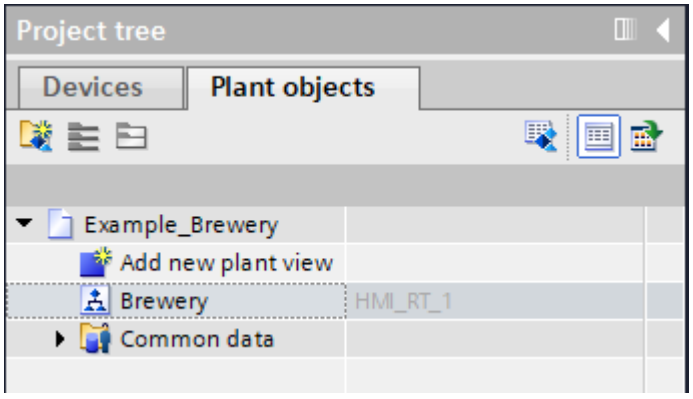
- A project has been created.
- An HMI device has been created.

Introduction

1. Under "Project tree > Plant objects", click "Add new plant view".
   An empty plant view is created.
2. According to our example, call the plant view "Brewery".
3. Assign the plant view to the HMI device.

Result

You have created the "Brewery" plant view and assigned it to the HMI device:



15.3.8.5 Example: Creating plant objects and plant object types

Task

You create plant object types from which you will create plant objects in the plant view in the next step. The plant objects represent the units and production lines of the brewery. These are the following elements in our example:

- Filling tank (unit)
- Conveyor (unit)
- Robot (unit)
- Bottling (production line)
- Packaging (production line)

Next, you define the "Communication driver" property of the interfaces.

Requirement

- The project is open.
- A SIMATIC S7-1500 PLC has been created.
- At least one program block is configured in the PLC.
- The "Brewery" plant view is created, assigned to an HMI device and open.

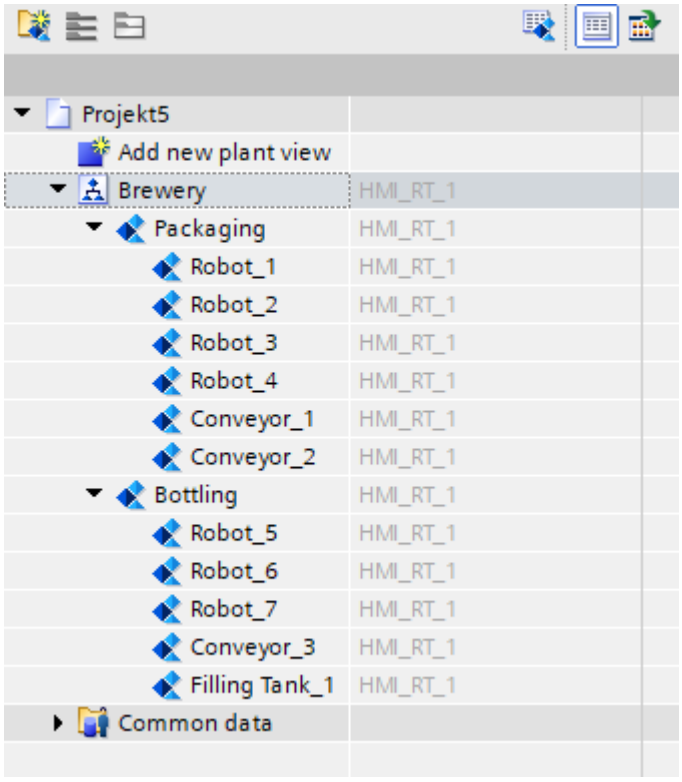
Procedure

1. To display the "Plant object types" task card, click the "Show plant object types" button under "Project tree > Plant objects".
2. Create a plant object type for each unit and assign the following names:
   - Conveyor
   - Robot
   - Filling Tank
   - Bottling
   - Packaging
3. Specify "SIMATIC S7 1200/1500" as communication driver for each plant object type.
4. As PLC tag, specify the function block that contains tags which match the unit.
5. Now create the following plant objects from the plant object types by dragging the respective plant object type to the plant view and adapting the name:
   - Packaging
   - Bottling
6. In the "Bottling" plant object, create the following lower-level plant objects by dragging the respective plant object type to the "Bottling" plant object:
   - Robot_1
   - Robot_2
   - Robot_3
   - Conveyor_1
   - Tank_1
7. In the "Packaging" plant object, create the following lower-level plant objects by dragging the respective plant object type to the "Packaging" plant object:
   - Robot_4
   - Robot_5
   - Robot_6
   - Robot_7
   - Conveyor_2
   - Conveyor_3
8. Under "Connection", assign each plant object the HMI connection and an instance data block as PLC tag.

Result

You have created plant object types for the units and production lines of the brewery. From the plant object types, you have created plant objects in the plant view.

The plant view should have the following structure:



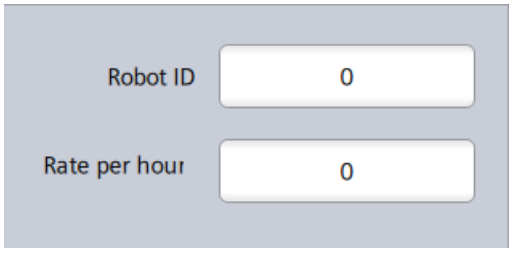
15.3.8.6 Example: Configuring screens for brewery production lines

Task

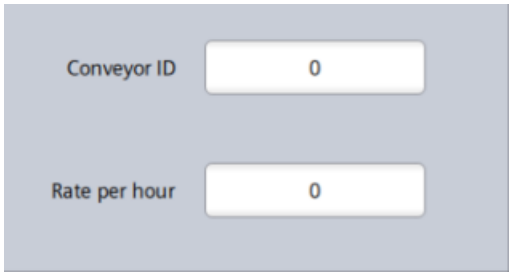
To visualize the plant objects, configure faceplate types for each plant object type.

Procedure

1. Under "Project tree > Plant objects", configure the screens for the "Bottling" and "Packaging" production lines.
   - Open the "Bottling" plant object and switch to the "Visualization" tab.
   - You press the "Add new screen" button to create a new screen.
   - Repeat the procedure for the "Packaging" plant object.
2. In the project library, you create a faceplate type for the plant object type "Robot" and release it.
   The faceplate could look as follows according to the example scenario:

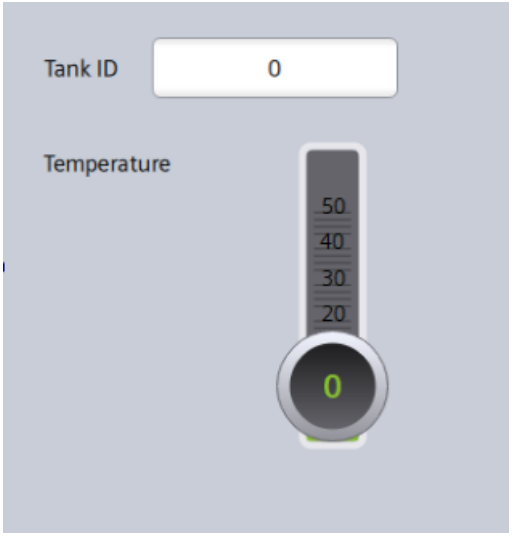


3. Under "Project tree > Plant objects", open the plant object type "Robot" and drag the created faceplate type to the "Save faceplates here" button in the "Visualization" tab.
4. Create faceplate instances for all the robots you need for the two production lines:
   - Open the "Bottling" screen.
   - Then drag all lower-level plant objects that are based on the "Robot" plant object type one after the other to the screen.
   - Repeat the procedure for the "Packaging" screen.
5. In the project library, you create a faceplate type for the plant object type "Conveyor" and release it.
   The faceplate could look as follows according to the example scenario:



6. Create faceplate instances for all the conveyors you need for the two production lines:
   - Open the "Bottling" screen.
   - Then drag all lower-level plant objects that are based on the "Conveyor" plant object type one after the other to the screen.
   - Repeat the procedure for the "Packaging" overview screen.

7. In the project library, you create a faceplate for the plant object type "Filling tank" and release it.
   The faceplate could look as follows according to the example scenario:



8. Under "Project tree > Plant objects", open the plant object type "Filling tank" and drag the created faceplate type to the "Drop faceplates here" button in the "Visualization" tab.
9. From the "Filling tank" plant object, create a faceplate instance for the filling container:
   - Open the "Bottling" overview screen.
   - Then drag the plant object that is based on the "Tank" type to the screen.

Result

You have successfully configured screens and faceplates for the plant objects of the brewery and can display the plant objects in the runtime.

See also

Configuring screens for plant objects (Page 7080)

Basic information on configuring screens (Page 7077)

15.3.8.7 Example: Configuring plant overview and companion controls

Task

To access the unit data directly from the "Overview" screen in runtime, configure a plant overview control in the overview screen.

To directly jump to the alarms of the "Filling Tank" plant object in runtime, create the alarm control as companion control to the "Plant overview" control. The alarm control is configured in the next step.

Procedure

1. Change to the project tree.
2. Under "Project tree > Devices", create an "Overview" screen.
3. Configure the "Plant overview" control in the "Overview" screen.
4. Also add the "Alarm control" and "Screen window" controls as companion controls to the "Overview" screen.

Result

You have configured an overview screen and added a screen window and an alarm control. Next, configure the alarm control.

15.3.8.8 Example: Configuring analog alarms for temperature monitoring

Task

The temperature of the beer brewing ingredients in the brewery must be strictly maintained. One of your tasks is to configure the temperature monitoring of the filling tank.

The following requirements apply to temperature monitoring of the filling tank:

- The setpoint for the temperature is 5 °C.
- When the temperature rises above 5 °C, the operator of the plant is notified.
- If the temperature reaches 7 °C, the operator is notified immediately. The operator has to confirm the notification.

Temperature deviations of the filling tank should be output on the HMI device.

You plan several escalation levels for the alarms to be output according to the requirements:

- Temperature is at 5 °C.
- Temperature is above 5 °C: An alarm that does not require acknowledgment is output.
- Temperature exceeds the critical temperature of 7 °C: An alarm that requires acknowledgment is output.

The temperature sensor of the filling tank returns analog values. Use these values to specify the triggers. The triggers determine when an alarm is triggered.

Requirement

- A trigger tag is configured for temperature monitoring, for example, "temperature".
- The "Filling Tank" plant object type is open.
- The "Analog alarms" editor is open.

Procedure for creating an alarm for exiting the setpoint range

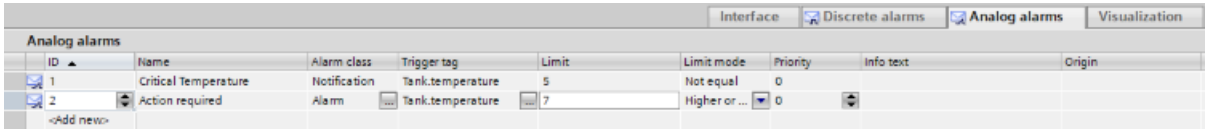
1. Create a new alarm and call it "Critical Temperature", for example.
2. Select the alarm class "Notification".
   Alarms of this alarm class do not require acknowledgment.
3. Select the trigger tag.
4. Define the trigger with a limit "5".
   This corresponds to the setpoint of 5 °C. Limits are always without units. The physical unit depends on the plant component which delivers the values.
5. Configure "Not equal" as the mode limit.
   When the value of the trigger tag is not equal to 5, an alarm is output.

Procedure for creating an alarm for exceeding the critical temperature

1. Create the "Action required" alarm.
2. Select "Alarm" as the alarm class.
   Alarms of this alarm class are displayed flashing in red on the HMI device and require acknowledgment.
3. Define the trigger with limit "7".
   This corresponds to a critical temperature of 7 °C.
4. Configure "Higher" as the mode limit.
   When the value of the trigger tag is higher than 7, an acknowledgeable alarm is output.

Result

You have configured alarms for the temperature monitoring:



See also

Configuring analog alarms for plant objects (Page 7091)

15.3.8.9 Example: Configuring the alarm control for fill level monitoring

Task

The configured alarms for temperature monitoring are to be output in an alarm control in runtime.

To filter by plant object alarms in the alarm control, configure a filter with the criterion "Area" with one of the following two conditions:

- "Equal" - only shows the alarms of the selected plant object in runtime.
- "Begins with" - shows the alarms of the underlying objects of the selected plant object in runtime.

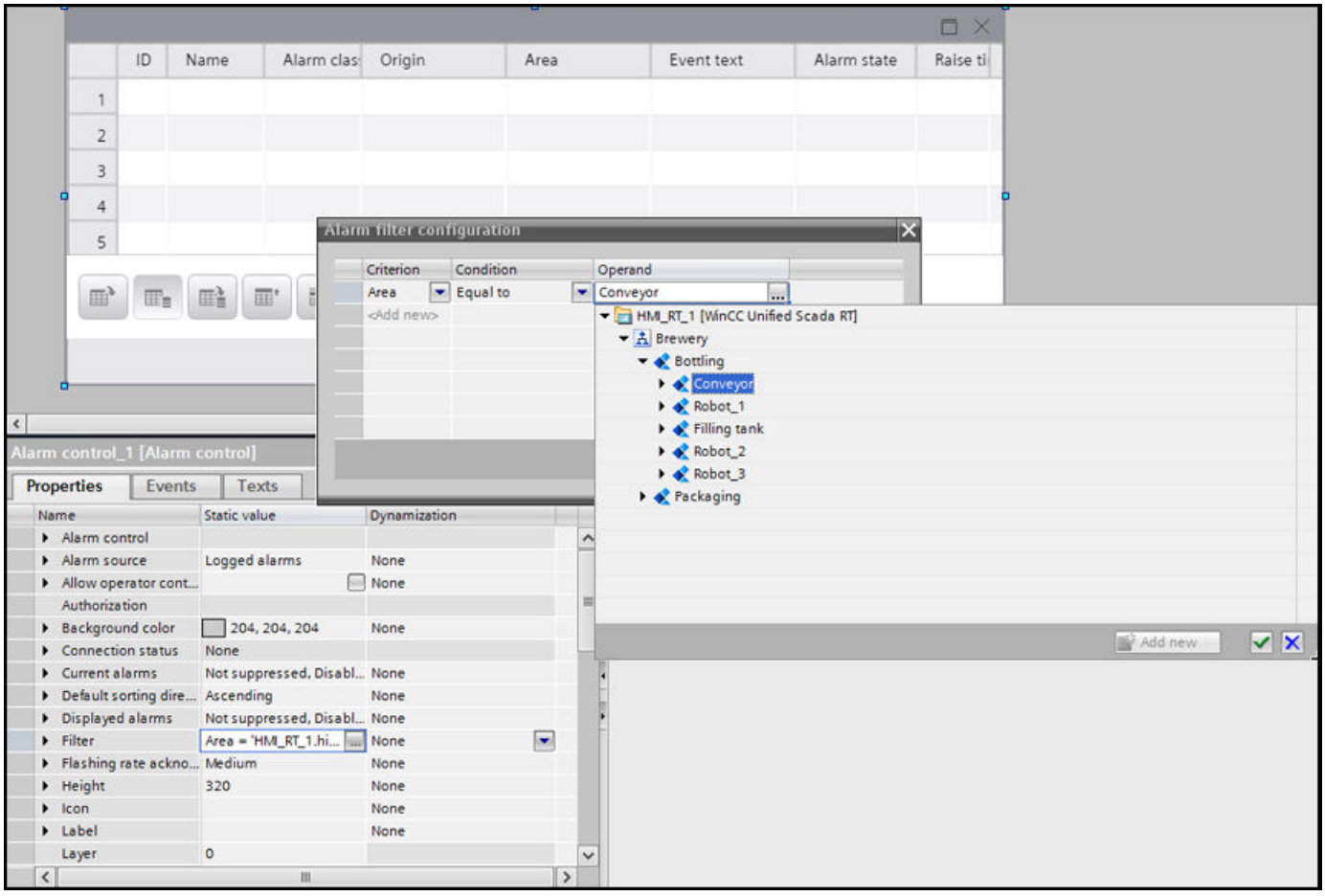
Requirement

- The "Overview" screen is created and open for editing.
- Configure a "Plant overview" control in the "Overview" screen.
- The "Alarm control" control is created as companion control of the plant overview.

Configuring a filter for the "Filling Tank" plant object

To filter by alarms of the "Filling Tank" plant object in the alarm control, configure a filter as follows:

1. In the Inspector window under "Properties > Filter", click in the "Static value" column. The "Alarm filter configuration" dialog box opens.
2. Select the "Area" criterion.
3. Select the condition "Equal".
4. Click on the selection list in the "Operand" column.
5. Select the "Filling Tank" plant object whose alarms are to be displayed in runtime.



Result

You have configured an alarm control for alarms of the "Filling Tank" plant object. The alarms of the plant object can be displayed in runtime.

15.3.8.10 Example: Configuring a trend view for temperature monitoring

Introduction

To visualize the tag values of the temperature monitoring in runtime, you have added a trend view in the "Overview" screen. You can display the current temperature of the "Filling Tank" plant object with the trend view.

Requirement

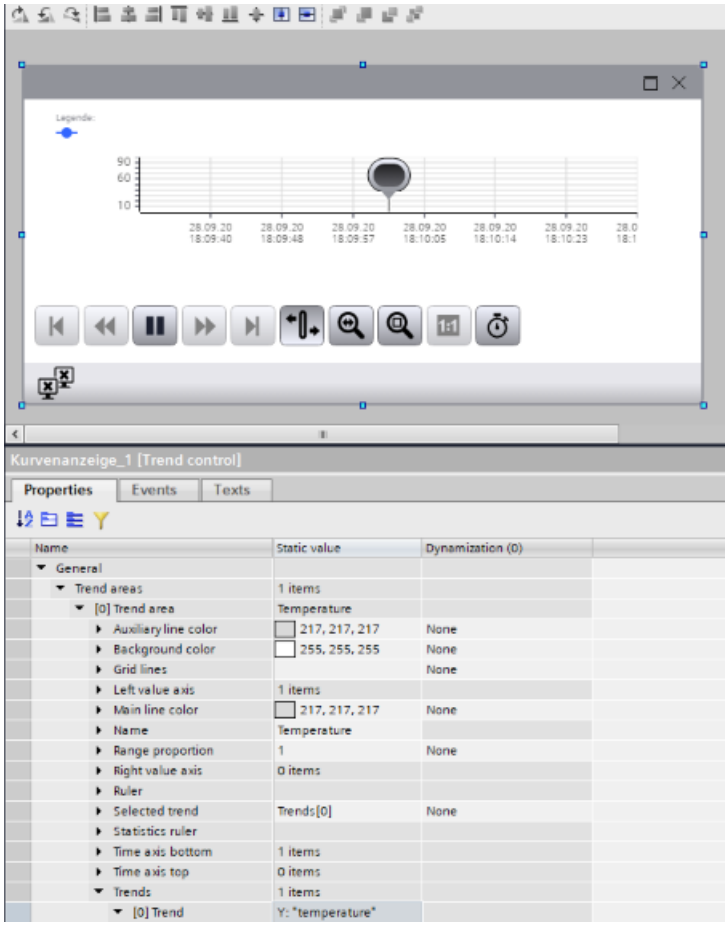
- The HMI tag for temperature measurement has been configured, for example "temperature".
- The "Overview" screen is open for editing.

Configuring the trend area and axes

1. Create a trend view in the "Overview" screen by dragging the control from the "Tools" task card to the screen.
2. Go to "Properties" and set the required height, width and position of the trend view.
3. Open the "Trend areas" group under "Properties".
   The index numbers of the trend area are displayed.
4. Expand the index number of the trend area.
   The properties of the trend area are displayed.
5. Define the colors for displaying the trend area and the reference lines.
6. Configure the time axis and value axis settings under "Bottom time axis" and "Left value axis".

Configuring trends

1. Go to "Properties > Properties > General > Trend areas > [0] trend areas > Trends" and click on the selection button in the "Static value" column.
   A dialog opens.
2. Click "Add" in the "Index" column.
   This adds another trend. Close the dialog.
3. Expand the index number of the trend [0]. The trend settings are displayed.
4. Specify the name of the trend under "Display name", for example "Temperature".
5. Under "Data source Y > Source", select the "temperature" tag in the "Static value" column.
6. Configure the line color for the trend, for example, blue.



Result

The trend control is now configured. In runtime, you monitor the temperature changes over time based on the trend.

Configure an additional value display if you want to evaluate the data of the trend control in runtime. You can also configure the value display as a "Ruler".

15.3.8.11 Example: Configuring the logging of production values

Task

You want to log the temperature values of the "Filling Tank" plant object to be able to use them later for controls, for example.

To this end, you create a logging tag for the data member you want to log. This logging tag is saved in the data log of the assigned device.

You can analyze the logged tag values directly in your project, such as in a trend view, or in another user program, such as Excel.

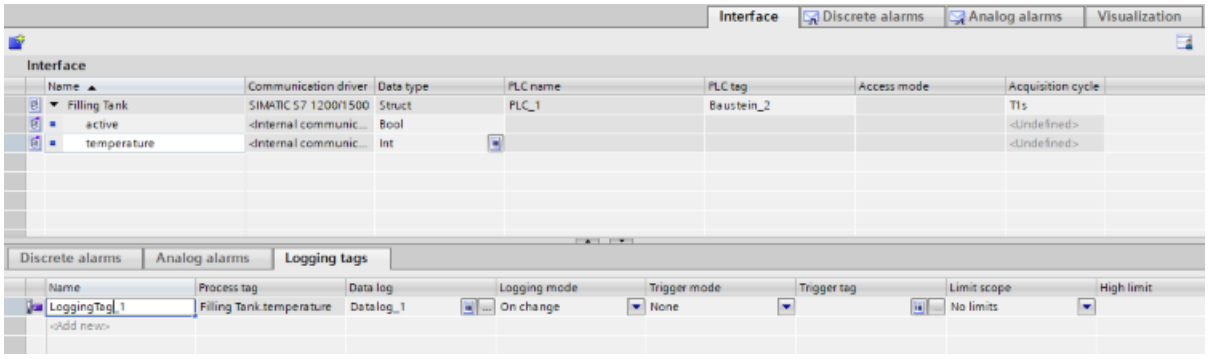
Logging tags are created for the plant object types. This means that the plant objects are automatically supplied with the logging tags of the plant object types.

Requirement

- The "Brewery" plant view has been created and assigned to a device.
- The "Filling Tank" plant object type is created, and the "temperature" member has been configured.

Procedure

1. Open the "Conveyor" plant object type.
2. Under "Interface", jump to the "Logging tags" tab in the middle part of the work area.
3. Under "Interface" select the "Rate" data member, for example, to log it.
4. Click "Add" under "Logging tags".



A logging tag is created.
The logging tag is linked to the tag. The data type of the logging tag corresponds to the data type of the connected tag.

5. Specify the logging mode, for example, "Upon change".
6. Define the tag trigger depending on the logging mode.
7. Define the limit values.
8. Define the smoothing.

Additional information on logging tags and logging is available under "Visualizing processes with Runtime Unified".

Result

You have created a logging tag that logs tag values for the defined period and within the parameters you have specified.

15.4 Visualizing plant objects in runtime

15.4.1 Displaying plant objects in runtime

Overview

Depending on your configuration, the following possibilities are available to you in runtime:

- Screen navigation via the plant model
- Analysis based on plant objects
- Filter alarm control by plant objects
- Display alarm status of a line and navigation to the alarm source
- Display the most frequently occurring alarms, filtered by plant object or plant object type
- Area-based access protection
- Determine the energy consumption of a line and compare with another line

You can display the configured plant hierarchy in runtime using the "Plant overview" control. If a screen window was configured as companion control for the "Plant overview", you can navigate between the screens of the plant objects and show them alternately in the screen window.

Display process data of the plant objects in a trend control. Switch between the following display modes directly in runtime:

- Device view and plant hierarchy
- Online values and log values

You can view alarms on plant objects in an alarm control.

See also

Operating "Plant overview" in runtime (Page 7114)

Display process data of the plant objects in a trend control (Page 7116)

Displaying alarms for plant objects in runtime (Page 7120)

15.4.2 Operating "Plant overview" in runtime

Introduction

Display the configured plant view in runtime using the "Plant overview" controls.

You use it to navigate to the plant objects within the plant structure and get an overview of your plant at one glance.

If you have configured screens or alarms for the lower-level plant objects and have linked them to the "Plant overview" control, display these screens and alarms in runtime.

If you have configured events for the "Plant overview" control and linked these to scripts, the scripts are called when the events occur.

An event can, for example, be linked to the operation of the buttons in the control.

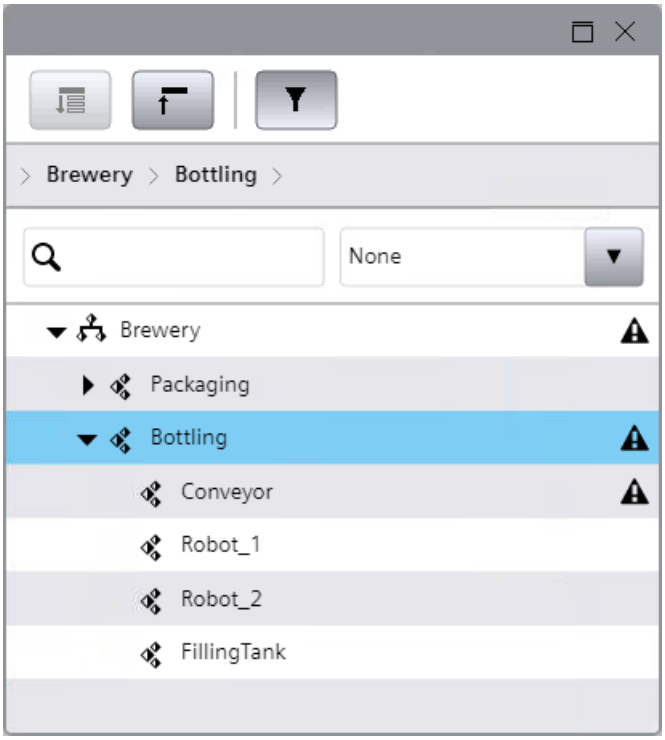
Requirement

- The plant view has been created and assigned to a device.
- The "Plant overview" control and the corresponding companion controls are configured in the screen of the assigned device.
- Runtime is active.

Procedure

The plant view is displayed in the "Plant overview" control.

1. To display all lower-level plant objects, click [[ICON]] "Expand all".
2. To display the configured screen or screen window for a selected plant object, click on the respective plant object in the "Plant overview" control.
3. In case of an alarm, the plant path of the alarm source is displayed in the "Range" column of the connected alarm view.
   The alarm icon only appears an alarm has actually occurred at the respective plant object. The alarm icon disappears again when the alarm is no longer present.



See also

Configuring screens for plant objects (Page 7080)

Displaying alarms for plant objects in runtime (Page 7120)

Display process data of the plant objects in a trend control (Page 7116)

Displaying plant objects in runtime (Page 7113)

15.4.3 Display process data of the plant objects in a trend control

Introduction

The process data or the logging data of the plant objects are displayed graphically in a trend control in runtime.

You switch between the following display modes in runtime:

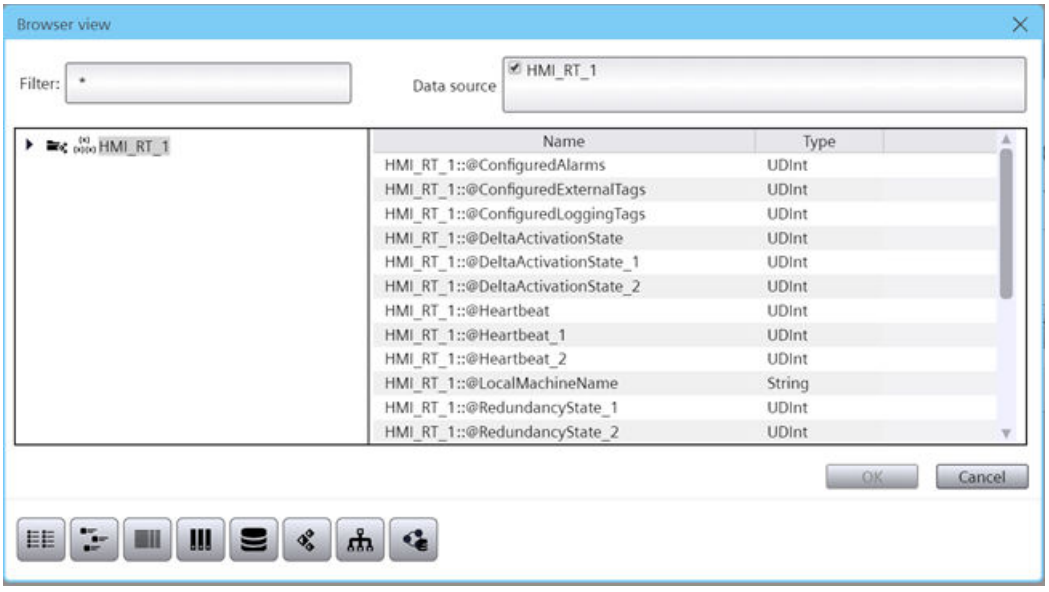
- Device view and plant hierarchy
- Online values and log values

Requirement

- The plant hierarchy has been created and assigned to a device.
- A trend control is configured in the screen by the assigned device.
- The "Select data connection" button is displayed in the control.
- The "Select context" button is displayed in the control.
- Runtime is active.

Display process data of the plant objects

1. Click "Select data connection" in the trend control.
   A "Selection of logs/tags" dialog opens.
2. To open the list of available tags, click "Tag".
   The "Browser view" dialog opens.



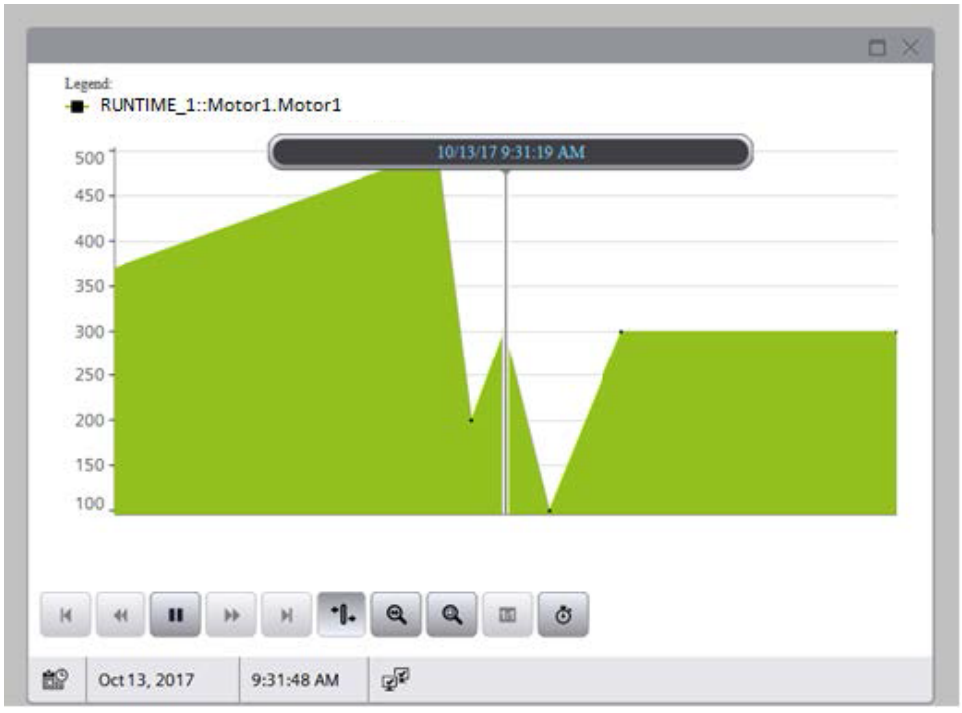
3. To jump to the plant view dialog, click on the "CPM" icon in the toolbar.
   The plant objects and the available data members for the plant objects are displayed.

Note

If you have assigned a descriptive display name for the trend when you configured the trend control, only the display name is shown in runtime. All plant objects are visible at a glance in the selection list.

4. Select the respective plant object whose process data you want to display in the trend control.

5. Select the data members that you want to display as trend in the trend control.
6. Confirm with "OK".
   The process values for the selected plant object are displayed in the trend control.



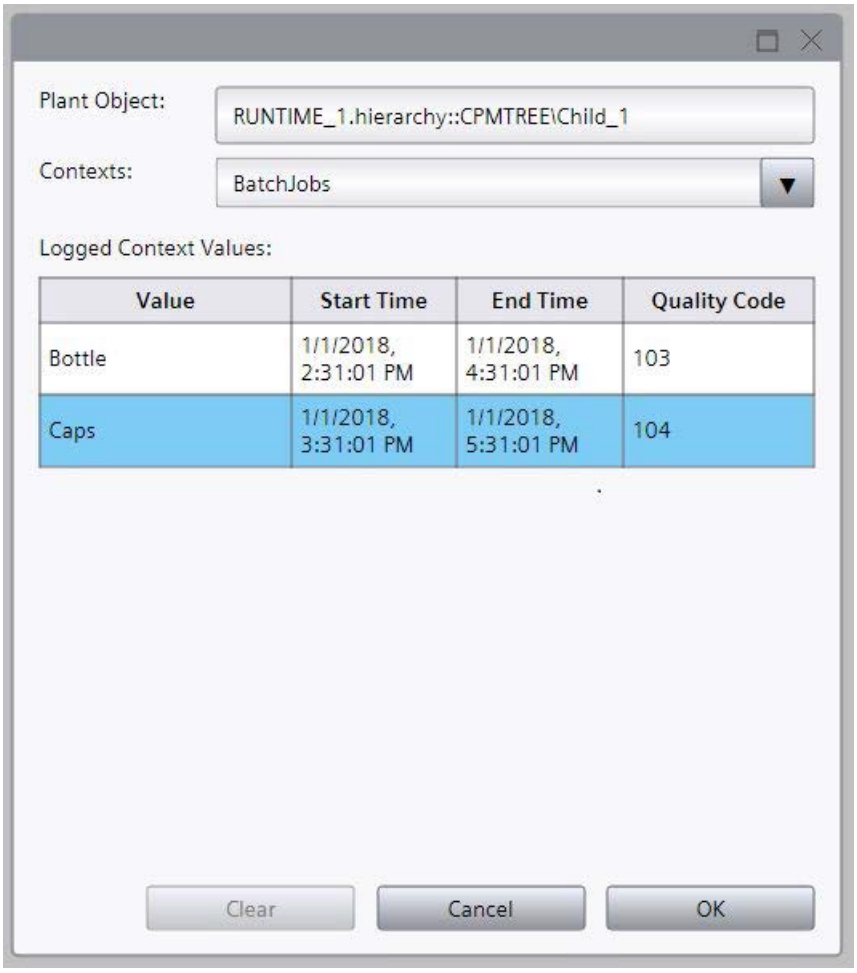
Display context data of the plant objects in a trend control

For analysis purposes, display the value range of the resulting data using the context data.

The evaluation is relevant, for example, in connection with WinCC Unified Performance Insight to analyze the effectiveness or the fault rate of the plant.

1. In the trend control, click "Select context".
   The "Trend context" dialog opens.
2. In the "Plant objects" selection list, select the respective plant object whose data you want to display in the trend control.
3. In the "Context" selection list, select the data assigned to the plant object.
   The list of the data appears under "Logged context values".

4. Select the value that you want to display.
5. Confirm with "OK".



The trend control displays the trends for the selected data.

See also

Operating "Plant overview" in runtime (Page 7114)

Displaying alarms for plant objects in runtime (Page 7120)

Configuring trend control for plant objects (Page 7085)

Displaying plant objects in runtime (Page 7113)

Contexts (Page 7063)

15.4.4 Displaying alarms for plant objects in runtime

Introduction

The "Alarm control" object displays alarms that occur during the production process in a plant.

Depending on your configuration, the following possibilities are available to you in runtime:

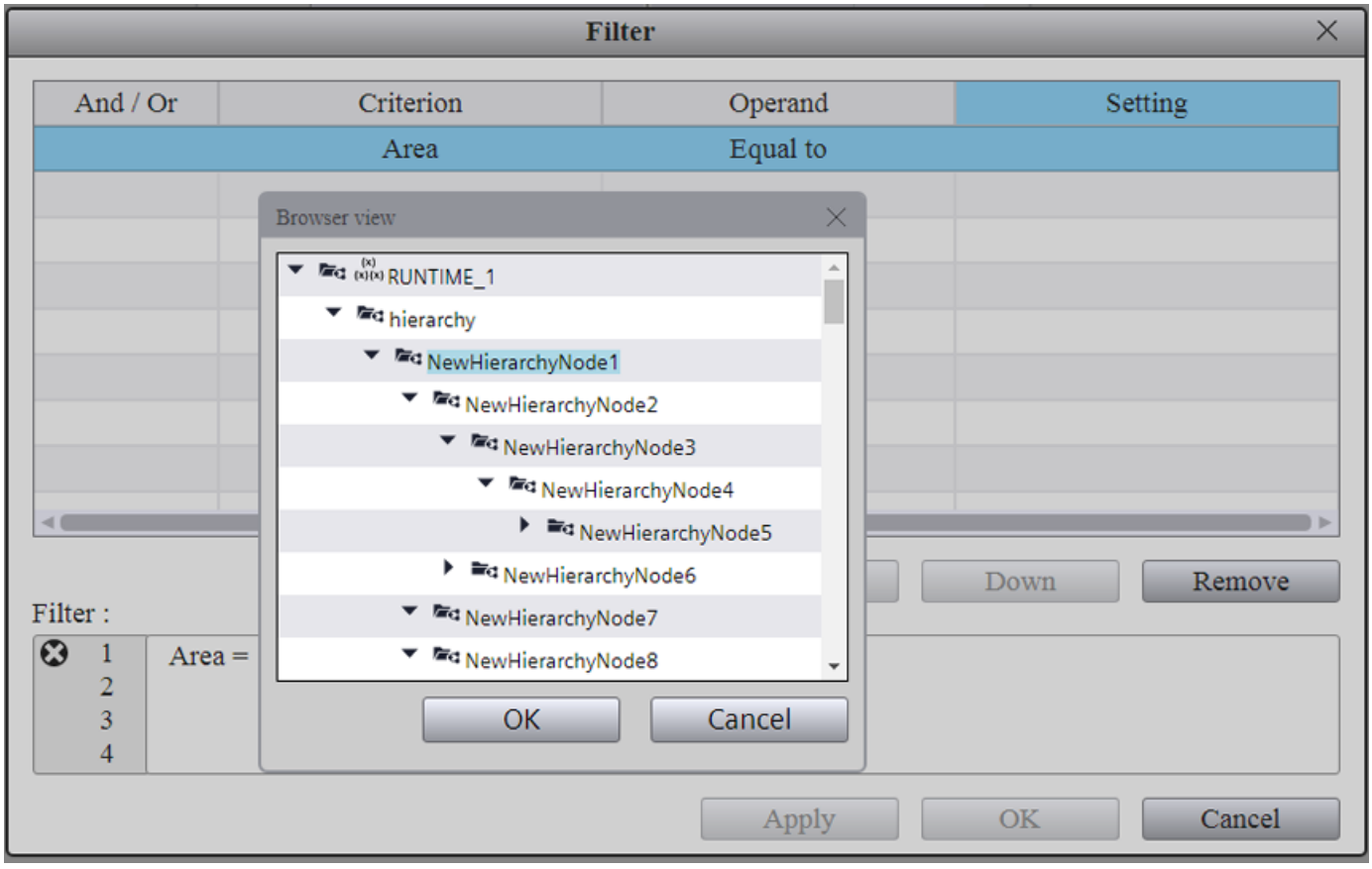
- Display hierarchy path of the alarm source
- Filter alarm control by plant objects
- Display alarm status of a line
- Navigate to the alarm source
- Display the most frequently occurring alarms, filtered by plant object or plant object type

Requirement

- The plant hierarchy has been created and assigned to a device.
- An alarm control with the filter "Range" is configured in the screen by the assigned device.
- Runtime is active.
- The alarm view has been configured.
- Runtime is active.

Filter alarms by plant objects

1. If you want to filter alarms by plant objects, click "Selection display" in runtime.
   The "Selection" dialog opens.
2. Under "Criterion", select the criterion "Range".
   All plant objects of the plant hierarchy are displayed.
3. Select the respective plant object to which you want to display the alarms.



Only alarms for the selected plant object are displayed in the alarm control.

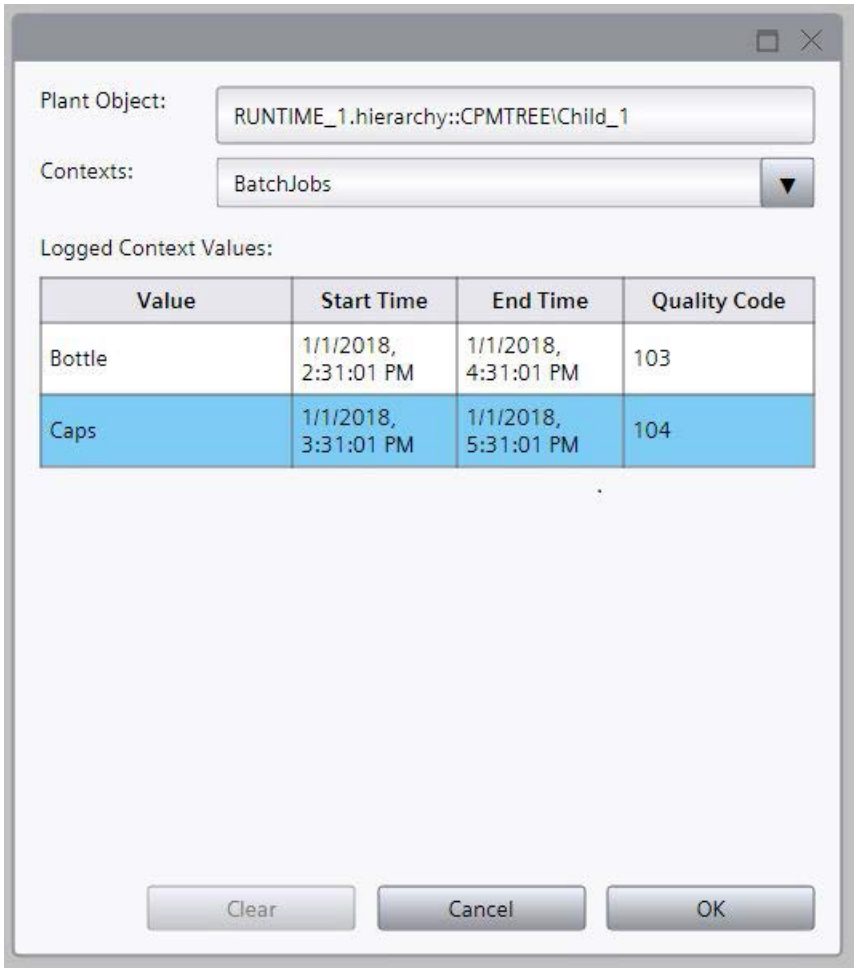
Display alarm context of the plant objects

For analysis purposes, display the value range of the resulting data using the context data.

The evaluation is relevant, for example, in connection with the WinCC Unified Performance Insight to analyze the effectiveness or the fault rate of the plant.

1. In the alarm control, click "Select context".
   The "Alarm context" dialog box opens.
2. In the "Plant objects" selection list, select the respective plant object whose data you want to display in the alarm control.
3. In the "Context" selection list, select the data assigned to the plant object.
   The list of the logging data appears under "Logged context values".

4. Select the value that you want to display.



5. Confirm with "OK".
   The alarm control displays the alarms that match your selection.

Note

Make sure that the filter settings match the setting of the alarm context.

If no alarms appear in the alarm control, check your settings by clicking "Selection display".

See also

Operating "Plant overview" in runtime (Page 7114)

Display process data of the plant objects in a trend control (Page 7116)

Configuring an alarm control for plant objects (Page 7084)

Configuring analog alarms for plant objects (Page 7091)

Configure discrete alarms for plant objects (Page 7088)

Displaying plant objects in runtime (Page 7113)

15.5 Options

15.5.1 Plant Intelligence Options

Overview

The Plant Intelligence options offer optional enhancements to the WinCC Unified Basic System. These can be combined freely in line with your requirements.

The options allow you to plan production processes and analyze and optimize the overall effectiveness of your plant. In addition, you can design flexible production processes and coordinate complex and interlinked production processes.

Plant Intelligence options

- WinCC Unified Performance Insight
  Define, calculate and analyze plant-specific key performance indicators (KPIs) for individual aggregates, machines or entire production lines in machine-oriented or line-oriented manufacturing plants.
- WinCC Unified Calendar
  Plan, configure and manage events and actions together in a shared calendar in WinCC and combine these with WinCC tags or scripts.
- WinCC Unified Sequence
  Control step-based and sequence-based processes, define the production steps of the production units and adapt the production processes flexibly in runtime.
- WinCC Unified Line Coordination
  Coordinate and supervise processes in the production line in your plant. Control and manage recipes, processes and jobs for the production of various end products.

Note

The Plant Intelligence options are successively released as add-on packages. To use the Plant Intelligence options, you require the relevant software packages and licenses.

You can find information on the licenses in the TIA Portal installation instructions in the section "Licensing of Plant Intelligence options".

Requirements

Please note the following requirements for using the options:

- SIMATIC WinCC Runtime Unified is installed.
- STEP7 Professional is installed.
- Plant Intelligence option, including license, is installed.
- The plant hierarchy is configured.
- License for the respective option is available.
- The configuration engineer has WinCC experience.

Compiling and loading 16

16.1 Basics

16.1.1 Overview

Introduction

To generate an executable runtime project from the configuration data of an HMI device, you must compile the project. You can then transfer the runtime project to the HMI device. This section explains the terms used in this context.

Project

In the context of compiling and downloading, the term "project" is used as follows:

- WinCC project: Contains the configuration data of one or more HMI devices in the engineering system
- Runtime project: Contains the compiled configuration data of an HMI device.

Runtime

In runtime, you execute the project in process mode.

A distinction is made depending on the HMI device:

- Runtime on a panel
  The runtime software for process visualization runs on the HMI device.
  Before you run a runtime project on a panel, you have to transfer the runtime project to the panel.
- Runtime on a PC
  WinCC Unified PC RT is the runtime software for process visualization.
  If Runtime has been installed on the configuration PC, you can execute the runtime project directly on the configuration PC.
  If you want to execute the runtime project on a different PC, you have to transfer the runtime project to the PC.

Compiling and loading

To compile a project means generating a runtime project from the WinCC project.

Downloading a project means transferring the runtime project from WinCC to an HMI device.

Simulation

With a simulation, you test the configuration, for example, configured internal tags or a screen change. You simulate the project on the configuration PC.

See also

Using WinCC version compatibility (Page 187)

16.1.2 Power Tags

Tags that have a connection to the PLC are called Power Tags.

Information on the Power Tags used is helpful to

- Know when the limits of the system are reached
- Be able to estimate the performance
- Be able to estimate the system load through communication

Information on Power Tags used

The number of Power Tags used is displayed in the Inspector window during the following operations:

- A device is compiled.
- Only changes for a device are compiled.
- A device is loaded.
- Only changes for a device are loaded.

16.1.3 Workflow

Introduction

This section provides an overview of the work steps from creating the WinCC project to displaying runtime on the HMI device.

They work with various applications:

- WinCC Engineering System in the TIA Portal
- WinCC Runtime system on a Unified PC or Unified Comfort Panel

- If the HMI device is Unified PC:
  - SIMATIC Runtime Manager
    You manage the Runtime projects in Runtime Manager.
    You can find additional information in the runtime help in the "SIMATIC Runtime Manager" manual.
  - WinCC Unified - Configuration
    In WinCC Unified - Configuration, you check and manage the settings for secure communication in Runtime.
- If the HMI device is a Unified Comfort Panel:
  - Control Panel
    In the Control Panel, you can start and stop the Runtime project or define a time interval after which the Runtime project starts automatically. You manage the transfer settings in the Control Panel.

Configuration in the engineering system

You start the configuration in the engineering system.

1. Create a project.
2. Add one of the following HMI devices:
   - PC General with the HMI application WinCC Unified PC
   - Industrial PC with the HMI application WinCC Unified PC
   - SIMATIC Unified Comfort Panel
3. Configure the required contents of the HMI device.
4. Check the runtime settings of the HMI device, especially the settings for secure communication.

Compiling and loading

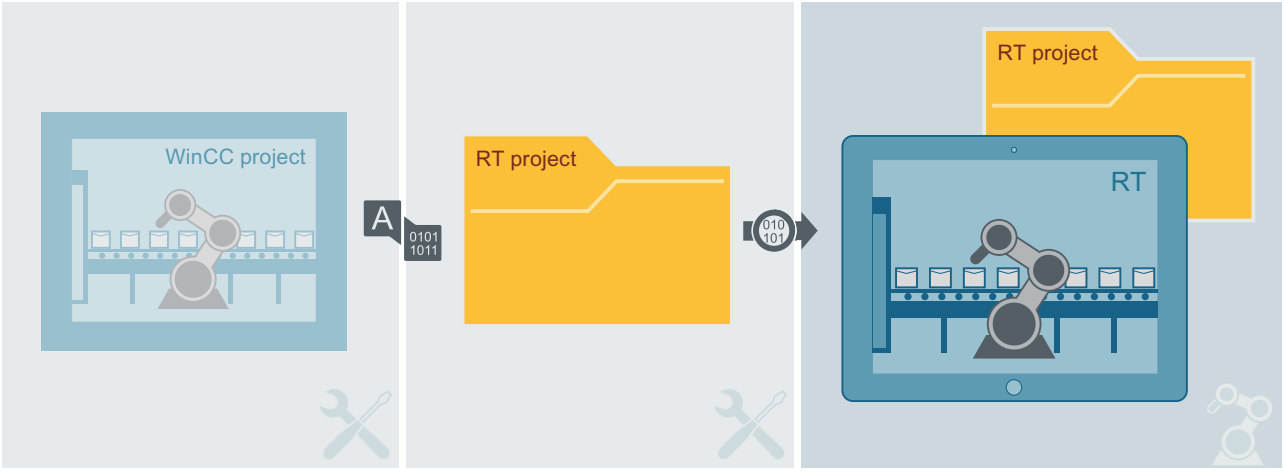
Then compile your WinCC project into a runtime project and download it to the target HMI device. Before you start the download, check to ensure that the settings for secure communication in the Engineering System and runtime are compatible.

1. Compile the WinCC project.
2. Download the runtime project directly to the HMI device:
   - When you first download the project, set up the connection to the HMI device before you start downloading.
   - When you download the project again, you can either download it completely or changes only.

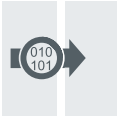
   For Unified PC, you can also copy the project onto an external storage medium and transfer it to the HMI device via the storage medium. You can download the complete project or download only changes.

Note

Not all changes can be loaded in Runtime with the option "Download to device > Software (only changes)". A list of changes and actions that require complete compiling and complete download can be found under "Restrictions in compiling and loading changes (Page 7132)".



The configuration data are compiled.



The runtime project is loaded.

Starting the runtime project

Before you start downloading the project, you specify in the "Load preview" dialog whether the project is started in runtime after the download.

Depending on the HMI device, you have the following options to start the project at a later time:

- Unified PC: To start the runtime of the project later, use the SIMATIC Runtime Manager.
- Unified Comfort Panel: Depending on the configuration, the project starts automatically after the delay time specified in the Control Panel.
  If the project does not start automatically, click "Start Runtime" in the Control Panel.

Displaying screens in runtime

Depending on the HMI device, runtime is displayed as follows:

- Unified PC: You can see Runtime in the browser. The user must be logged on in runtime.
- Unified Comfort Panel: Runtime is displayed on the HMI device after the start.

See also

Loading project encrypted (Page 7130)

Loading project unencrypted (Page 7131)

16.1.4 Secure communication

Introduction

You download your Runtime projects encrypted and password protected using secure communication.

The following components are involved:

- Engineering System on the configuration PC: You manage the "Encrypted transfer" option under "Runtime settings" of the created HMI devices.
- Runtime on the HMI device
  - Unified PC: You manage the "Secure download" option during the installation of the runtime software or in the "WinCC Unified - Configuration" application.
  - Unified Comfort Panel: You set the password for encrypted transfer to the HMI device in the Control Panel under "Service and Commissioning > Transfer".
- SIMATIC Runtime Manager (only for Unified PC): The password is specified in the settings of the Runtime Manager under "General > Secure connection".
  If the "Secure download" option is enabled in Runtime, data exchange is only possible if the identical password is stored in the Runtime Manager.
  You can find more information in the runtime help in the "SIMATIC Runtime Manager" manual.

  Note

  If the passwords in the Runtime Manager and Runtime do not match, the project cannot be managed in the Runtime Manager.

Runtime is encrypted and password-protected when the "Secure download" option is selected. The data exchange with Runtime is only possible in the engineering system if authentication was successful. The password must be available in the runtime settings of the HMI device and it must match the password in runtime.

Password

The password must meet the following criteria:

- A length of 8 to 120 characters
- At least one special character
- At least one number
- At least one lowercase letter
- At least one uppercase letter

Settings for encrypted transfer in the Engineering System

By default, the encrypted transfer is enabled in the runtime settings of the HMI device.

In the runtime settings of the HMI device under "General > Encrypted transfer", make the following settings:

- Entering a password in the engineering system
- Allow transfer of initial password via unencrypted loading
- Deactivate encrypted transfer

Multiple runtime projects in one runtime

When you download multiple runtime projects to one runtime with WinCC Unified PC, assign the same password in the runtime settings of the created HMI devices in the engineering system. This password must match the password stored in runtime.

See also

Loading project encrypted (Page 7130)

Loading project unencrypted (Page 7131)

16.1.5 Loading project encrypted

Introduction

If you want to use secure communication, the encrypted transfer must be enabled in the runtime settings of the HMI device. The password must be entered once and confirmed.

Depending on the configuration of runtime, we distinguish between two cases:

- "Secure download" is configured in runtime.
- "Secure download" is not configured in runtime.

"Secure download" is configured in runtime

If the password was configured in WinCC Unified Configuration or during the installation of the runtime software, the option "Allow transfer of initial password via unencrypted loading" is not needed.

If the password in the engineering system matches the password configured in runtime, the project is loaded encrypted.

Note

Deviating passwords

If the passwords in the runtime settings of the HMI device and in runtime are different, you will have the opportunity to enter the password stored in runtime in the "Load preview" dialog before you download the project. If the input was successful, the password in runtime is replaced by the password in the runtime settings of the HMI device.

The project is loaded encrypted.

To use the Runtime Manager, update the password in the settings of the Runtime Manager.

"Secure download" is not configured in runtime

If the option "Secure download" is not configured in runtime, you can activate the option "Allow transfer of initial password via unencrypted loading":

- The runtime project is initially loaded unencrypted.
- Secure download is activated.
  The password entered in the runtime settings of the HMI device is written to the runtime configuration.
  The change can be verified in WinCC Unified Configuration.
- The runtime project is loaded encrypted for all subsequent download operations. The option "Allow transfer of initial password via unencrypted loading" is no longer taken into account.
- To use the Runtime Manager, update the password in the settings of the Runtime Manager.

If the option "Allow transfer of initial password via unencrypted loading" is not enabled, the runtime project is not loaded. An error message is displayed in the "Load preview" dialog.

See also

Secure communication (Page 7129)

Loading project unencrypted (Page 7131)

16.1.6 Loading project unencrypted

If encrypted transfer is disabled in the runtime settings of the HMI device, a distinction is made between two cases, depending on the configuration:

- "Secure download" is configured in runtime.
  The project is not loaded.
  An error message is displayed in the "Load preview" dialog.
  The encrypted transfer must be enabled in the runtime settings of the HMI device.
- "Secure download" is not configured in runtime.
  The project is loaded unencrypted.

See also

Secure communication (Page 7129)
Loading project encrypted (Page 7130)

16.1.7 Restrictions in compiling and loading changes

Compiling and loading changes

Many changes to the configuration can be compiled and loaded in Runtime with the options "Compile > Software (only changes)" or "Download to device > Software (only changes)". After some changes or actions, however, the project must be completely compiled or completely loaded.

NOTICE
Option to compile and load changes is lost Please note the following instructions for compiling and loading changes: • A dialog is often displayed when the option to compile only changes is about to be lost. The change can be confirmed or rejected. When you reject the change, the option to compile and load changes is retained. • When you use the "Undo" button to undo a change that requires complete compiling or loading, the project must still be completely compiled or loaded. • For the relevant changes and actions, an alarm is displayed in the Inspector window when the option to load changes is already lost. The project must be downloaded completely.

Complete compilation or loading may be necessary under the following circumstances:

- Creating, changing or deleting configuration data
- Actions that directly affect the compiling and loading (e.g. starting a simulation)
- System-dependent conformity errors (for example, through the installation of an update that must update the resultant compilation data or owing to the compilation crashing)

Note

Complete compilation always requires complete loading.

The following changes or actions require complete compilation or complete loading:

Area of the change or action	Complete compilation required	Complete download required
Tags	• All changes to the configuration of complex tags of the Array or Struct data type	• Renaming simple tags • Changing the data type for simple tags
Styles	• Creating styles • Changing styles • Deleting styles	-
Connections	• Creating connections • Changing connections • Deleting connections	-
Cycles	• Creating cycles • Changing cycles • Deleting cycles	-
Plant objects and plant views	• Creating plant objects and the plant view • Changing plant objects or the plant view • Deleting plant objects or the plant view	-
Logging	• Creating the first data log by creating system tags • Changing a data log • Deleting a data log • Creating an alarm log • Changing an alarm log • Deleting an alarm log	• Changing a log name • Changing the path when using the backup • Changing the data type of the tag to which the logging tag is linked. • Changing the logging mode • Changing the compression mode • Changing the source for using compression
Audit	• Changing the Audit Trail	• Activating the GMP-compliant configuration • Deactivating the GMP-compliant configuration • Changing the Audit Trail name • Changing the Audit Trail storage medium • Changing the Audit Trail storage directory
Unified Collaboration	• All changes in the Runtime settings under "Collaboration"	-
Language & font	• All changes in the runtime settings under "Language & Font"	-
Identification	• Renaming the WinCC Unified PC RT: Renaming changes the Runtime ID. The Runtime ID can be viewed in the Runtime settings under "General".	-

Area of the change or action	Complete compilation required	Complete download required
UMC (User Management Component)		• Creating a local user • Changing a local user • Deleting a local user • Creating a role • Changing a role • Deleting a role • Switching between local and central user management
Alarms	-	• All changes to alarm classes • Changing the alarm class for discrete alarms or analog alarms • Changing the name, alarm class or the range for alarms from OPC UA A&C
Resource lists	-	• Renaming text lists • Renaming Graphic lists
Simulation	-	• When you load the project in runtime mode and then load the same project in simulation mode, the simulation must be loaded completely. • When you load the project in simulation mode and then load the same project in runtime mode, the project must be loaded completely.
Reporting		• Changing the Runtime settings for reporting
Dynamic SVG types		• Manual assignment of a version number to a type

See also

Compiling a project (Page 7145)

16.2 Unified Comfort Panel

16.2.1 Specifying runtime settings

16.2.1.1 Introduction

Before you compile and download a project, update the runtime settings of the HMI device. You specify the Runtime languages, for example, or activate collaboration.

To edit the runtime settings for an HMI device, open "Runtime settings" in the project tree below the HMI device.

16.2.1.2 General

Identification

The runtime ID is the unique identification of a runtime project. Based on the runtime ID, you can determine whether a project has already been downloaded to the HMI device.

Encrypted transfer

To download the project encrypted, the same password must be stored in the runtime settings and in runtime. Alternatively, transfer the password unencrypted during the initial download.

Screen

Under "Screen", you specify the start screen and the style of the HMI device.

The start screen is displayed after runtime start.

Select the screen resolution. The setting of the HMI device is used by default.

Note

Display of a changed start screen

You have defined a start screen in the project and started runtime. When you define another start screen and download only changes to the device, the last active screen is displayed in runtime.

After reloading the project, refresh the screen in Runtime.

See also

Secure communication (Page 7129)

16.2.1.3 Alarms

Controller alarms and diagnostics

Note

WinCC only supports controller alarms of a SIMATIC S7-1500. WinCC only supports controller alarms that are automatically updated by the central alarm management in the PLC.

One or several HMI connections to a PLC are shown in "Controller alarms".

You can manage the following options:

- "Display classes":
  You filter the controller alarms via display classes. You specify for each connection which display classes are displayed on the HMI device.
- "Automatic update":
  Controller alarms are automatically updated on the HMI device.
  Requirement: The option "Central alarm management in the PLC" must be enabled.
- "System diagnostics":
  System diagnostic alarms can be displayed in runtime.
  Requirement: The "Automatic update" option must be enabled.
- "Security events":
  Security events can be displayed in runtime.
  Requirement: The "Automatic update" option must be enabled.

State texts

You specify the state texts of alarms in the runtime settings. The state texts are displayed in runtime in the alarm control.

Specify the state texts in other languages under "Languages & Resources > Project texts".

See also

Configuring the display of security events (Page 795)

Configuring state texts of alarms (Page 733)

Configuring the display of system diagnostic alarms (Page 767)

Filtering controller alarms via display classes (Page 748)

16.2.1.4 Services

SMTP communication

SMTP communication enables automatic sending of e-mails when events occur.

Specify the port for SMTP communication.

16.2.1.5 Language & font

Runtime language and font selection

You configure project languages that are available as runtime languages for the respective device. You also define the order in which the languages are switched.

At Runtime start, the language that has the lowest number in the "Order" column under "Runtime settings > Language & font" in the TIA Portal is used. You change the order with  .

The fixed font 1 is always provided for the respective HMI device.

When you enable the option "Enable for logging" for a language, alarm texts are logged in the respective language. To keep the size of the log relatively small, log only alarm texts in the required languages.

See also

Editing log contents with scripts and system functions (Page 850)

Enabling the runtime language (Page 246)

16.2.1.6 Remote access

Collaboration

General settings

- Select the "Enable collaboration" check box.

Identification

The following information must be unique for all devices participating in a collaboration:

- System ID:
  The system ID must be unique for each device participating in Unified Collaboration, as this ID is used for communication between the devices.
- Collaboration name:
  Assign a collaboration name or select "Generate collaboration name automatically".
- IP address / Host name

Connect actively to

A list of all HMI devices available for collaboration with system ID and IP address / host name is displayed. You select the HMI devices to which the current HMI device provides collaboration.

Note

Collaboration is enabled in both directions

The HMI device that you selected under "Connect actively to" can display screens of the current HMI device and vice versa. Collaboration is always enabled in both directions, even if you only have the connection activated for one collaboration device.

See also

Defining collaboration settings (Page 7572)

Web Client

Web client configuration

When you activate access via web client, you can access the runtime of the Unified Comfort Panel from any browser.

Smart Server

Smart Server configuration (only available for Unified Comfort Panel)

When you activate the Smart Server, you enable remote access to the Unified Comfort Panel via the Smart Client application.

Users

You set passwords for up to two users and enable the users for remote access to the control panel of the Unified Comfort Panel.

Communication

- You enable or disable the use of self-signed certificates by selecting or clearing the check mark.
- Port configuration
  You enable or disable the automatic port configuration by selecting or clearing the check mark.
  If you want to specify a manual port for access from desktop applications, enter the port number.

See also

WinCC Smart Server (Page 7606)

16.2.1.7 Storage system

Introduction

You specify the storage locations of the following databases:

- Database for tag persistency
- Database for data logging
- Database for alarm logging

NOTICE
Different storage locations cannot be used for Unified Comfort Panel When you specify a different storage location for a log in the "Logs" editor other than the main database storage location defined in the runtime settings of the HMI device, the log cannot be used.

You have the option of saving parameter set types on external storage media. If the databases for data logs or alarm logs and the parameter set types are stored on the same storage medium and the storage medium is changed while runtime is running, the parameter set types can be affected.

Note

Use different storage media for parameter set types and logging.

Database type

Unified Comfort Panel supports the "SQLite" database type.

Database storage location for tag persistency

You can specify tag persistency for internal tags. The last values of the persistent tags are used after Runtime has been started.

A separate database is used for tag persistency. The values of the tags are available again after restarting runtime or restarting or switching off the HMI device.

Specify the storage location of the database:

- Off
- SD-X51
- USB-X61
- USB-X62

If no storage medium is connected to the respective interface, an alarm is displayed in the alarm control.

If the log databases and the database for tag persistency are stored on the same storage medium and the storage medium is changed while runtime is running, tag persistency can be affected.

Note

Use different storage media for tag persistency and logging.

Location of the main database for data logging

Specify the storage location of the databases for data logging:

- Off
- SD-X51
- USB-X61
- USB-X62

You have the option of specifying a path on the storage medium.

If no storage medium is connected to the respective interface, an alarm is displayed in the alarm control.

Location of the main database for alarm logging

Specify the storage location of the databases for alarm logging:

- Off
- SD-X51
- USB-X61
- USB-X62

You have the option of specifying a path on the storage medium.

If no storage medium is connected to the respective interface, an alarm is displayed in the alarm control.

See also

Specify tag persistency (Page 636)

Storage locations of logs (Page 843)

16.2.1.8 Settings for tags

You have the option of synchronizing HMI tags with the connected PLC tags. The position of a data value in the structure of a data block is thus mapped in the HMI tag name. If necessary, the PLC name is set as a prefix. You synchronize the tags in the "HMI tags" editor.

You specify before the synchronization whether and under which conditions the names are synchronized.

Synchronization of the name of the PLC tag in the engineering station

To avoid conflicts within complex tag types, configure how the delimiters of the path specification from STEP 7 are replaced in WinCC during name synchronization:

- Replace delimiters
  Depending on your selection, the delimiters of all hierarchy levels are replaced during synchronization.
- Replace invalid characters
- PLC prefix
  The PLC name is set as a prefix to the HMI tag name. You set the option for each HMI connection.

Note

Duplicate tag names

If the generated tag name is already in use, a number is added in parentheses, e.g. Datablock_1_Static_2{1}(1).

Example

The "PLC1" controller contains the structured data block "DB1". The "Db1.a[1].b.c[3]" data block element is used in a picture. Depending on your settings, the HMI tag name is generated as follows:

Selected option	HMI tag name
PLC prefix	Plc1.Db1.a[1].b.c[3]
Replace dot and parenthesis with ; ( )	Db1;a(1);b;c(3)
Replace dot and parenthesis with _ { } PLC prefix	Plc1_Db1_a{10}_b_c{3}

See also

Synchronizing tags (Page 640)

16.2.1.9 Good Manufacturing Practice

Traceability and therefore the documentation of production data is becoming increasingly important in many sectors such as the pharmaceutical industry, the food and beverage industry, and the related mechanical engineering industry.

Therefore, sector-specific and cross-industry standards have been developed for the electronic documentation of production data.

The most important set of regulations is the FDA guideline 21 CFR Part 11 for electronic data records and electronic signatures issued by the FDA, the US Food and Drug Administration. In addition, different EU regulations apply, such as EU 178/2002, depending on the industry.

Requirements for production systems in these industries have been developed on the basis of 21 CFR Part 11 and the corresponding layout to comply with GMP (Good Manufacturing Practice). They are also required for other industries.

The following main requirements are derived from these directives and regulations:

- Creation of an Audit Trail or operating trace in runtime
  Based on this document, it is possible to trace the user who carried out the operator action on the machine at what time.
- Important process steps must also be assigned to a clear responsibility, for example, via an electronic signature.

GMP (Good Manufacturing Practice)

1. Enable GMP-compliant configuration.
2. If necessary, select a text list entry that contains the reason for the GMP-relevant tags.

16.2.1.10 User management

You specify whether you are using a local or a central user management. By default, the use of the local user management is specified in the engineering system.

Local users are only valid for this project.

You manage central users in the TIA User Management Component (UMC).

User administration configuration

- When you activate the local user management, you use the users and user roles that you have created under "Security settings > Users and roles" for management.
- When you activate the central user management, the users, user roles and their rights are applied from the TIA User Management Component (UMC). To access the UMC, you must specify the server address and the server ID.

See also

Local and central user management (Page 6889)

16.2.1.11 OPC UA server

General

OPC is a standardized manufacturer-independent software interface for data exchange in automation engineering. OPC UA is the technology succeeding OPC. OPC UA is platform-independent and supports different protocols as communication medium.

To work with OPC UA in WinCC Unified, the OPC UA server must be enabled in the TIA Portal in the Runtime settings of the HMI device.

Read/write tags and register tags/alarms

When you enable the "Operate as OPC UA server" option in the HMI device, the protection for unauthorized internal and external access is downgraded.

- Enable the "Operate as OPC UA server" option.
  A security note is displayed.

After enabling the option, all other settings of the OPC UA Server will become available.

Alarms and Conditions

- To display alarm conditions in the address range of the server, select the option "Enable Alarms and Conditions on the OPC UA server".
- To disable or acknowledge alarms on the OPC UA Client, for example, select the option "Allow operation of alarms on the OPC UA Client". To enable this option, the "Enable Alarms and Conditions on the OPC UA server" option must be enabled.

Options

General

Define the following settings:

- Port
  Default value: 4890
  Do not use a port number that is already assigned to another application.
- Maximum session timeout (s)
  Default value: 600000 s
- Maximum number of OPC UA sessions
  Default value: 100

Subscriptions

Define the following settings:

- Minimum publication interval (ms)
  Default value: 100 ms
- Maximum number of monitored items
  Default value: 0

Security

Secure connection

Security policies

CAUTION
Reduced security When the option "No OPC UA Server Security" is enabled, any OPC UA client can connect to the OPC UA server regardless of the following settings.

The following section contains a list of all security policies available on the server.

- Activate the required security policies.

User authentication

Guest authentication

- To allow access by anonymous users to the OPC UA server, enable the option "Enable guest authentication".
  An authentication by means of user name and password is not required.
  Security is restricted to the degree that you determine by assigning rights to this user.

Authentication by means of user name and password

- To allow access by users with user name and password to the OPC UA server, enable the option "Authentication with user name and password".
  If access to the OPC UA server is to require the user name and password, the user must be assigned the role "HMI Administrator". The "HMI Administrator" role has the system-defined "OPC UA - read and write access" function right. The settings made must then be synchronized with the user management in runtime.

16.2.1.12 Layers

Default names of the layers

Adjust the default names of the layers.

16.2.1.13 Reporting

Report system

For Unified Comfort Panels with device version V18, you make the settings for Reporting here.

"Enable Reporting"	You can create report templates based on this project or project report jobs and generate reports in Runtime only if reporting is enabled.	
"Storage location for the Reporting database"	The Reporting database stores the actions and settings made in Runtime in the "Reports" control.	
	"Storage location"	Select a storage medium.
	"Folder"	Enter the path to a folder of the storage medium. Use an existing folder. Use the Linux notation. The Reporting database is stored in the folder. Default folder: The project folder
"Storage location for reports"	Select the local main storage location for the generated reports.	
	"Storage location"	Select a storage medium.
	"Folder"	Enter the path to a folder of the storage medium. Use an existing folder. Use the Linux notation. The reports are stored in the folder. Default folder: "`media/simatic/<storage medium>/Reports`"

Note

Devices with a device version lower than V18

Unified Comfort Panels with a device version of lower than V18 always have the same settings:

- Reporting is enabled.
- The storage locations are predefined:
  Storage location for reports: `media/simatic/X51/Reports` folder on the SD card inserted in the panel.
  Storage location of the reporting database: The respective project folder under `media/simatic/X51` on the SD card plugged into the panel.

You cannot change the settings.

16.2.2 Compiling a project

Basics

Compilation results in a runtime project that is executable on the respective HMI device.

A project can be compiled explicitly or implicitly:

- Explicit: The compilation is started manually. Compile the entire project or only changes. Explicit compiling is described in the following section.
- Implicit: The project is downloaded without previous manual compilation.
  The project is compiled in the background even as you are configuring it in WinCC.
  This reduces the compilation times before downloading the project.

Note

Changes to S7 blocks are not automatically compiled in the background. If you are using HMI tags in the project that are connected to PLC tags, you should also compile all changed S7 blocks before you compile the HMI device.

Requirement

- A project is open.

Compiling changes of a project

1. Select an HMI device in the "Project tree".
2. There are two ways to start compiling the changes:
   - Press the "Compile" button in the toolbar.
   - Select "Compile > Software (only changes)" from the shortcut menu.

Compiling a project completely

1. Select an HMI device in the "Project tree" area.
2. Select "Compile > Software (rebuild all)" from the shortcut menu.

Note

When a project was compiled completely, the project must also be downloaded completely afterward.

Compiling projects of multiple HMI devices simultaneously

1. Select the HMI devices using multiple selection in the project tree.
2. To compile the changes to the projects, select the "Compile" button in the toolbar. Alternatively, select "Download to device > Software (only changes)" from the shortcut menu.
3. To compile the projects completely, select "Download to device" > "Software (rebuild all)" from the shortcut menu.

Result

The configuration data of the selected HMI devices is being compiled. If errors occur during compilation, the errors are shown in the Inspector window under "Info > Compile". Correct the errors and recompile the project.

Load the compiled project.

See also

Restrictions in compiling and loading changes (Page 7132)

Simulate Unified Comfort Panel (Page 7205)

Downloading projects (Page 7147)

16.2.3 Downloading projects

16.2.3.1 Basics for downloading projects

Introduction

During downloading, the compiled project is transferred to an HMI device. Only one project can be in runtime on an HMI device at a time.

Before the download, use the "Load preview" dialog to determine whether existing data is retained and whether logs are reset. The following data can be retained or overwritten:

- Runtime values:
  Tag values, active alarms and user data
- Logs:
  Data logs, alarm logs, Audit Trails and context logs

The HMI device name entered in the project tree is used for PROFINET communication. The use of the name corresponds to the default setting of the PROFINET interface of the HMI device. For devices with more than one PROFINET interface, the name of the IE CP is automatically added to the device name with a separating period. The name is written to the HMI device during download. If a device name for PROFINET communication has already been entered in the HMI device, it will be overwritten.

You can find additional information about these settings in the information system in the "Assigning a device name and IP address" section.

When TIA Portal detects incompatible Runtime versions during online loading, you have the option in the "Load Preview" dialog to install an image with a compatible, installed Runtime version to the Panel before the download.

NOTICE
Changing the installed Runtime version deletes all data from the HMI device. Data is deleted on the target system when you change the installed Runtime version. When the operating system is updated, the project, parameter sets and user administration on the HMI device are deleted. Before updating the operating system, save the data on the HMI device, if required. When the operating system is updated with a reset to factory settings, all data on the HMI device is deleted and all settings in the control panel are reset to the factory settings.

The compilation of the project is checked before downloading and missing content is compiled. This ensures that the latest version of the project is always downloaded.

When you download a project again, you can decide whether you want to download only changes or the complete project. If you want to download only changes, a compilation of changes must be possible beforehand.

Note

No data loss when loading is interrupted

Existing data on the HMI device is only deleted when the transfer is complete.

Runtime ID

At the start of the configuration, each project receives a runtime ID which is transferred to the HMI device during downloading. If you have already downloaded a project, the download process recognizes the project using the Runtime ID. If the HMI device name is changed in the configuration, the Runtime ID also changes.

Note

Existing runtime projects on the HMI device

If a runtime project with the same Runtime ID is already available on the HMI device, the project is overwritten. In this case, the options "Software (only changes)" and "Software (all)" are available for loading.

If a runtime project with a new runtime ID is downloaded to the HMI device, the existing runtime project is replaced by the new project. In this case, only the option "Software (all)" is available for the download.

In both cases, existing project data on the HMI device is overwritten.

Controlling the transfer behavior on the HMI device

You enable the transfer on the HMI device in the Control Panel under "Transfer". When the transfer is activated, the project can be downloaded.

Note

Disable transfer after commissioning

Disable the transfer after the commissioning phase so that no project can be loaded inadvertently.

An inadvertent transfer mode can trigger unwanted responses in the plant.

In order to restrict access to the transfer settings and thus avoid unauthorized changes, enter a password in the Control Panel.

For more information on the transfer settings, refer to the documentation of the HMI device.

Transferring Runtime add-ons in WinCC

Projects may contain Runtime add-ons in the form of controls or CSP (Communication Support Packages). These Runtime add-ons are automatically transferred with the project.

See also

Complete reloading of a project (Page 7152)

Compiling a project (Page 7145)

Initial download of a project (Page 7150)

Secure communication (Page 7129)

Restrictions in compiling and loading changes (Page 7132)

Simulate Unified Comfort Panel (Page 7205)

Compiling and loading with team engineering (Page 7157)

Basics on version compatibility (Page 187)

16.2.3.2 Initial download of a project

Introduction

The first download of a project is different from any subsequent download processes:

- The connection to the HMI device must be set up before the download.
- The project is always downloaded completely during the first download.

Note

Runtime is stopped during a complete download

The project running in runtime is stopped when a project is executed in runtime while you are completely downloading the project.

Requirement

- The project has been compiled without errors.
- A user is configured.
- The HMI device is connected to the configuration PC.
- The Control Panel is started on the HMI device.
- The protocol by which the project is loaded is set on the HMI device in the Control Panel under "Settings".
- The transfer is activated on the HMI device.

Editing connection parameters before download

1. Select the HMI device in the project tree.
2. Select "Online > Extended download to device" in the menu.
   The "Extended download" dialog opens.
3. Configure the interface.
4. Click "Connect" and load the project.
   The Runtime project is downloaded with changed connection parameters.

Options when initially loading a project

Runtime values: Tag values, active alarms and user data

Select whether you want to keep the current values or reset the values to the start values.

Select "Reset to start values" at the time of initial loading.

Reset logs: Data logs, alarm logs, Audit Trails and context logs

Select whether you want to reset all logs or no log in Runtime.

Initial loading of a project

1. Select the HMI device in the project tree.
2. In the toolbar, select the "Download to device" button or select "Download to device" > "Software (all)" from the shortcut menu.

   Note

   When you select "Download to device" > "Software (only changes)" from the shortcut menu, the project is still downloaded completely during the initial download.

   The "Extended download" dialog opens.
3. Configure the interface. Make sure that the settings match the transfer settings in the HMI device:
   - Select the protocol used, for example, Ethernet.
   - Configure the relevant interface parameters on the configuration PC.
   - Make any interface-specific or protocol-specific settings required on the HMI device.
4. Click "Connect".
   The connection is established and a dialog is displayed.
5. Select "Load".
   The compilation of the project is checked and missing content is compiled.
   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.
6. Check the displayed default settings and change the settings as necessary:
   - Specify whether runtime should start on the target system after the download.
   - Select "Reset to start values".
   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".
7. Click "Download".

Result

The project is downloaded to the selected HMI device with the runtime extensions it contains.

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

On completion of the successful download of the project, you can execute it on the HMI device.

Note

No data loss when loading is interrupted

Existing data on the HMI device is only deleted when the transfer is complete.

See also

Loading projects of multiple HMI devices simultaneously (Page 7155)

Specify tag persistency (Page 636)

Basics for downloading projects (Page 7147)

Secure communication (Page 7129)

Compiling a project (Page 7145)

16.2.3.3 Complete reloading of a project

Options when reloading a project

Runtime values: Tag values, active alarms and user data

Select whether you want to keep the current values or reset the values to the start values.

With the "Keep selected" option, you can specify which values you want to keep.

- Current values of tags and pending alarms
- Current user management data

Reset logs: Data logs, alarm logs, Audit Trails and context logs

Select whether you want to reset all logs or no log in Runtime.

Requirement

- The project has been compiled without errors.
- The project has been downloaded at least once before.

Reloading a project

1. Select the HMI device in the project tree.
2. Select "Download to device > Software (all)" from the shortcut menu.
   The compilation of the project is checked and content that has not been compiled is compiled.
   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.

Note

Runtime is stopped during a complete download

The project running in runtime is stopped when a project is executed in runtime while you are completely downloading the project.

3. Check the displayed default settings and change the settings as necessary:
   - Specify whether runtime should start on the target system after the download.
   - Specify whether tag values, active alarms, and user data are retained.
     The setting is only accepted when you have selected "Start runtime".
     To retain internal tags, the persistency must be enabled in the settings of the respective tag.
   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".
   - Specify whether the IDs of objects in the Engineering System and their relevant Runtime data should be synchronized.
4. Click "Download".

Note

To prevent the users created in the user management from being overwritten in runtime by the complete download of the project, activate the "Keep current user administration data in runtime" option.

When this option is selected, role assignments and function rights from the user management of the engineering system are loaded to Runtime, but not user data such as user name and password.

Result

The project with the Runtime add-ons it contains is downloaded to the selected HMI devices.

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

On completion of the successful download of the project, you can execute it on the HMI device.

Note

No data loss when loading is interrupted

Existing data on the HMI device is only deleted when the transfer is complete.

See also

Updating the operating system of the HMI device (Page 7167)

Error messages during loading of projects (Page 7159)

Basics for downloading projects (Page 7147)

16.2.3.4 Download changes only

Introduction

When you only download changes to a project, the relevant project must be executed in runtime. Runtime is not closed when loading.

Requirement

- The project has been compiled without errors.
- The project has been downloaded at least once before.
- A compilation of changes must be possible or have been executed.
- The project that contains the changes is being executed in runtime.

Procedure

1. Select the HMI device in the project tree.
2. Select "Download to device" in the toolbar.
   Alternatively, select "Download to device > Software (only changes)" from the shortcut menu.
   The compilation of the project is checked and content that has not been compiled is compiled.
   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.
3. Click "Download".

Result

The project with the Runtime add-ons it contains is downloaded to the selected HMI devices.

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

Runtime continues to be executed.

Note

No data loss when loading is interrupted

Existing data on the HMI device is only deleted when the transfer is complete.

See also

Basics for downloading projects (Page 7147)

Restrictions in compiling and loading changes (Page 7132)

16.2.3.5 Loading projects of multiple HMI devices simultaneously

Requirement

- Multiple HMI devices are configured.
- The individual projects have been compiled without errors.
- A user is configured.
- The HMI devices are connected to the configuration PC.
- The Control Panel has started on the Unified Comfort Panels.
- The protocol by which the project is loaded is set on the Unified Comfort Panel in the Control Panel under "Settings".
- "Automatic" or "Manual" is set as the transfer mode on the Unified Comfort Panels.

Connecting HMI devices to the configuration PC

1. Select an HMI device in the project tree.
2. Select "Online > Extended download to device".
   The "Extended download" dialog opens.
3. Configure the interface. Make sure that the settings match the transfer settings in the HMI device:
   - Select the protocol used, for example, Ethernet.
   - Configure the relevant interface parameters on the configuration PC.
   - Make any interface-specific or protocol-specific settings required on the HMI device.
4. Click "Connect".
   The connection is established and a dialog is displayed.
5. To connect additional HMI devices and load several projects at the same time, select "Cancel".
6. Repeat steps 1 to 5 for additional HMI devices.

Loading projects

1. Select the HMI devices using multiple selection in the project tree.
2. To download the changes to the projects, select the "Load" button in the toolbar.
   Alternatively, select "Download to device > Software (only changes)" from the shortcut menu.

3. To download the projects completely, select "Download to device" > "Software (all)" from the shortcut menu.
   The compilation of the projects is checked and content that has not been compiled is compiled.
   The result of the compilations is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed. All selected projects are listed in the dialog.

   Note

   Runtime is stopped during a complete download

   The project running in runtime is stopped when a project is executed in runtime while you are completely downloading the project.

4. Check the displayed default settings and adjust the settings for each device:
   - Specify whether runtime should start on the target system after the download.
   - Specify whether tag values, active alarms, and user data are retained.
     Only available if you have selected "Start runtime".
   - Specify whether all logs are reset in runtime.
     Only available if you have selected "Start runtime".
5. Click "Download".

See also

Initial download of a project (Page 7150)

Basics for downloading projects (Page 7147)

16.2.3.6 Using external storage medium

Introduction

If you cannot establish a direct connection from the configuration PC to the HMI device, load the compiled runtime project onto an external storage medium. For example, use a USB flash drive or SD card.

As soon as you have connected the external storage medium to your HMI device, transfer the project to the HMI device.

Requirement

- An HMI device has been created.
- The project has been compiled without errors.

Loading project to external storage medium

1. Jump to the "Devices" tab in the project tree.
2. Double-click on

3. "Add user-defined card reader" in the "Card reader/USB storage" folder.
   A selection dialog opens.
4. Select a target directory to save the project.
5. Drag and drop the folder of the HMI device (e.g. "HMI_1 [<Device type>]") to the added folder.
   Alternatively, use copy and paste.
   The project is checked. If the project has contents that have not yet been compiled, a compile is performed.
   The "Load Preview" dialog opens.
6. Select the options for loading.
7. Click "Load" to confirm.

Result

Your project is stored as a compressed ZIP folder in the directory "[<Target directory>]\Simatic.HMI\RT_Projects". The file name is made up of the name of the HMI device, the project name and the time stamp, for example "HMI_RT_1[Project1] - Full 2020-04-03 - 14.51.41.zip".

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

You can find information about loading the project from the external storage medium to the Unified Comfort Panel in the operating instructions for the Unified Comfort Panel.

16.2.4 Compiling and loading with team engineering

16.2.4.1 Compiling and loading with team engineering (overview)

Introduction

You can compile and download to an HMI device in the server project view, in the exclusive session and the local session.

Note

Unified objects cannot be selected in a local session

To edit an object using Multiuser Engineering, it must first be "selected". Only objects marked for check-in can be transferred into the server project after editing.

Unified objects cannot be marked in a local session. Changes to these objects are not applied to the server project during check-in.

You can edit unmarked objects in the server project view.

Basics

The options for compiling and downloading in the server project view, in the exclusive session or the local session are no different from the options in a single-user project. The most recent project is always compiled or loaded from the currently active view.

In principle, you can execute all commands for compiling and downloading in multiuser engineering and exclusive engineering:

- "Software (rebuild all)"
- "Compile > Software (only changes)"
- "Software (all)"
- "Download to device > Software (only changes)"

Rules

The following rules apply to compiling and downloading in multiuser engineering and exclusive engineering:

- The project that was changed in a local session always remains local and is not uploaded to the multiuser server.

  Note

  A Unified project that was created or changed in a local session cannot be saved in the multiuser server project.

  Use the local session to test your configuration. When you update your local session, all changes to Unified objects are overwritten by the server project.

- Only projects that were created or changed in the server project view or in the exclusive session can be saved in the multiuser server project.

See also

Compiling in the server project view (Page 7158)

Downloading projects (Page 7147)

16.2.4.2 Compiling in the server project view

Basics

Compiling and downloading of projects in the server project view and the exclusive session is no different from compiling and downloading in a single-user project.

While you are compiling a project in the server project view or the exclusive session, the server project is blocked. Other users cannot edit the server project in the meantime. The compiled runtime project is saved with the WinCC project on the central server. Blocking

the server project ensures that the configuration data and the runtime project remain synchronized.

Note

When you compile and save in the server project view or in the exclusive session, other users then obtain the Runtime project you have updated along with the WinCC project when they "refresh" their local session. Other users do not have to recompile the changes you have made after an update.

See also

Compiling and loading with team engineering (overview) (Page 7157)

16.2.5 Error messages during loading of projects

Possible problems during the download

When a project is being downloaded to the HMI device, status messages regarding the download progress are displayed in the output window.

Problems arising during the download of the project to the HMI device are usually caused by one of the following errors:

- Wrong version of operating system on the HMI device
- Incorrect settings for downloading to the HMI device
- Incorrect HMI device type in the project
- The HMI device is not connected to the configuration PC.

Download failures and possible causes and remedies are listed below.

The download is canceled due to a compatibility conflict

Possible cause	Remedy
Conflict between versions of the configuration software used and the operating system of the HMI device	• Synchronize the operating system of the HMI device with the version of the configuration software. • To update the operating system of the HMI device, select the command "Update operating system" in WinCC in the menu "Online > HMI device maintenance" or in the "Load preview" dialog. You can also use ProSave. For additional information, refer to the operating instructions for the HMI device.
The configuration PC is connected to a wrong device, e.g. a PLC.	• Check the cabling. • Correct the communication parameters.

Project download fails

Possible cause	Remedy
Connection to the HMI device cannot be established (alarm in the output window)	• Check the physical connection between the configuration PC and the HMI device. • Check that the HMI device is in transfer mode. Exception: Remote control

The configuration is too complex

Possible cause	Remedy
The configuration contains too many different objects or options for the HMI device selected.	• Reduce the project size.

See also

Reducing the project size (Page 7162)

16.2.6 Starting runtime

Introduction

You can start the project in runtime as soon as you have downloaded the project to the HMI device. By default, the project is started automatically on the HMI device.

The project settings defined in the "Runtime settings" of the HMI device are activated when the project is started in runtime.

Note

Response of runtime when the HMI device is restarted

When the HMI device is restarted, the project is automatically restarted even if the project was stopped before the restart.

Note

Closing Runtime automatically

If the transfer is activated on the HMI device and a transfer is started on the configuration PC, the running project is automatically terminated.

The HMI device then automatically switches to "Transfer" mode.

After the commissioning phase, disable the transfer so that the HMI device does not inadvertently switch to the "Transfer" mode.

"Transfer" mode can trigger unwanted reactions in the plant.

To restrict access to the transfer settings and thus avoid unauthorized changes, enter a password for access to the Control Panel.

Requirement

- The project has been downloaded to the HMI device.
- The Control Panel has been started.

Starting runtime on an HMI device

The Control Panel is displayed when the HMI device is switched on. Depending on the configuration, the loaded project starts automatically after the defined delay time.

If the project does not start automatically, click "Start" in the Control Panel.

Refer to the documentation for the HMI device for additional information on startup of projects.

Result

Runtime is started on the HMI device.

16.2.7 Reducing the project size

Introduction

When loading a large project to an HMI device, the HMI device may not have enough memory for the project.

Options for reducing the project size

There are several ways to reduce the size of the project and save space:

- Reduce the number of available runtime languages
  Check if all selected runtime languages are needed.
  If necessary, you can disable the languages that you do not need under "Runtime settings > Language & Font > Runtime language and font selection".
- "Software (rebuild all)"
  In order to optimize the project data and to clean up obsolete changes, compile the entire project using the "Compile > Software (rebuild all)" command from the shortcut menu of the HMI device.
- Reduce the number of fonts loaded
  Check if the number of user-defined downloaded fonts can be minimized.
  To save memory space, use fewer font groups for the configuration.
- Reduce the size of the graphics
  High-resolution graphic objects require a lot of memory and cause long loading times. They also reduce performance in Runtime.
  Check the size of the graphics that you use in the project. If necessary, reduce the size of the graphics by reducing the resolution or choose a higher compression format without noticeable loss of quality for the project graphics. Note the display resolution of the target device and the size in which the graphic object is displayed on the display of the target device.
  Select appropriate graphic formats for your screens: Use PNG images for drawings that are not vector graphics and JPEG images for photos.

See also

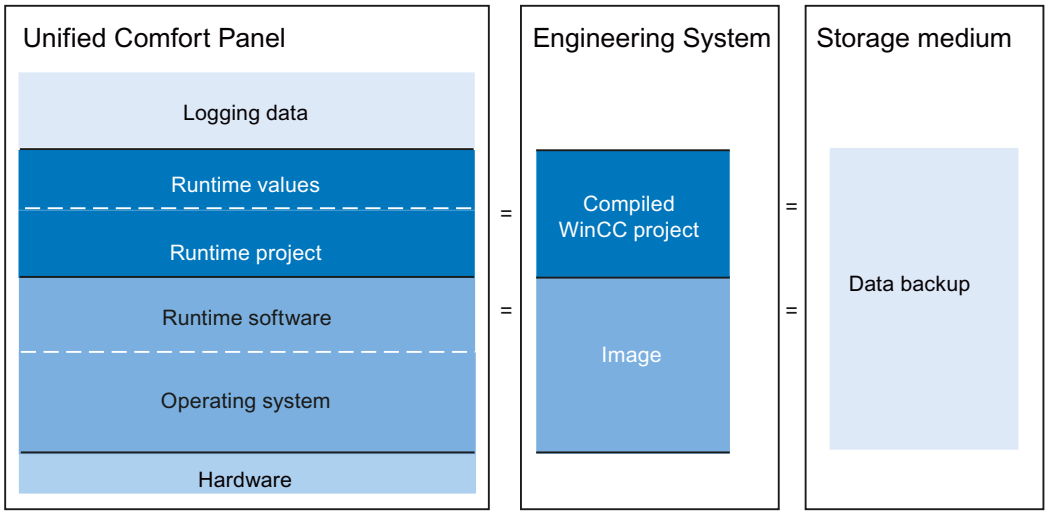
Complete reloading of a project (Page 7152)

16.2.8 Maintenance of the HMI device

16.2.8.1 Overview of the service for Unified Comfort Panels

Structure

The following figure shows the software components of an HMI device and their relation to the engineering system.



Logging data

Tags and alarms can be saved to logs. The log databases are stored on an external storage medium.

Runtime values

The runtime data is created during operation of the plant and stored on the HMI device. This data includes, for example, parameter sets and data for the user administration.

Runtime project

The runtime project contains the compiled configuration data for an HMI device.

Operating system and runtime software

The operating system of the HMI device is provided together with the runtime software in the form of an HMI device image. Suitable HMI device images are supplied with each WinCC version. Depending on the configuration, download the appropriate image along with the runtime project to the HMI device as required.

Firmware and hardware

The HMI device is delivered with preconfigured firmware and hardware.

16.2.8.2 ProSave

Introduction

The "ProSave" service tool is included in the WinCC installation. The ProSave functions are accessed in WinCC with the menu "Online > HMI Device maintenance".

Functional scope

ProSave offers numerous functions for data transfer between the configuration PC and HMI device:

- Backing up and restoring data of the HMI device
- Updating the operating system of the HMI device
- Communication settings (transferred from WinCC)

16.2.8.3 Data backup of the HMI device

Introduction

Data backup is used to create a backup of the data on the HMI device, e.g. before the update of the operating system. You can restore the backed-up data at a later time.

If an HMI device is connected to the configuration PC, you can back up and restore the data of the HMI device from the configuration PC using WinCC.

Alternatively, you can back up the data to an external storage medium supported by the HMI device. If the HMI device is networked, you can also back up the data to a server.

Scope of data backup

You have the option of performing a complete backup.

The following components are saved with this:

- Runtime
- Firmware
- Operating system
- Configuration
- Parameter sets
- User management
- Options

A backup includes several files. The master file has the extension ".brf". The number of additional files is variable, these files have the file name of the master file and a consecutive number (".0", ".1", ".2", ...) as extension.

Note

Scope of data backup

The selected content of the flash memory is saved during data backup. Alarm logs and data logs are stored on the external storage medium and are therefore not saved using the "Save" function. If necessary, back up the contents of the memory card separately.

Note the following for a complete backup and restore of the dataset:

- A full backup includes all options installed. All data of the option that are still present after "Power off" are saved.
- If the data is completely restored, all data previously on the device, including the operating system, will be irrevocably deleted.
- If the recovery process cannot be completed due to a power failure or an interrupted data connection, for example, the HMI device starts in maintenance mode and must be reset to factory settings.

Use an interface with high bandwidth, such as Ethernet, to back up and restore data via WinCC.

See also

Backing up and restoring data of the HMI device (Page 7165)

16.2.8.4 Backing up and restoring data of the HMI device

Note

Use the restore function for project data only on HMI devices which were configured using the same configuration software.

Requirement

- The HMI device is connected to the configuration PC.
- The HMI device is selected in the project tree.
- If a server is used for data backup: The configuration PC has access to the server.

Backing up data of the HMI device

To back up the HMI device data, follow these steps:

1. Select "Online > HMI device maintenance > Save" in the menu.
   The "Create backup" dialog box opens.
2. Select the type of the PG/PC interface and the target device, and click "Create".
   The "SIMATIC ProSave" dialog box opens.
3. Under "Data type", select the data to backup for the HMI device.
4. Enter the name of the backup file under "Save as".
5. Click "Start Backup".

This starts the data backup. The backup operation takes some time, depending on the connection selected.

Restoring data of the HMI device

To restore the data of the HMI device, follow these steps:

1. Select "Online > HMI device maintenance > Restore" in the menu.
   The "Restore backup" dialog box opens.
2. Select the type of the PG/PC interface and the target device, and click "Load".
   The "SIMATIC ProSave" dialog box opens.
3. Enter the name of the backup file under "Open...".
   Information about the selected backup file is displayed under "File information".
4. Click "Start Restore".

This starts the restoration. This operation takes some time, depending on the connection selected.

Backup/restore from the "Backup/Restore" dialog in the Control Panel of the HMI device

The "Backup / Restore" function is enabled for SD memory cards and USB memory media.

For more information, refer to the operating instructions of the HMI device.

See also

Data backup of the HMI device (Page 7164)

16.2.8.5 Updating the operating system

Introduction

If the image of an HMI device has a version status that does not match the configuration, update the image of the HMI device. The version of the image matches the installed Runtime version.

Update the operating system and the runtime software of the HMI device using the installed Runtime version. Depending on the protocol used, you may be prompted to run an automatic update of the installed Runtime version while loading the project.

Loading will then continue.

Update of the installed Runtime version

To update the installed Runtime version, connect the HMI device to the configuration PC. If possible, use the interface providing the highest bandwidth for this connection, e.g. Ethernet.

See also

Updating the operating system of the HMI device (Page 7167)

16.2.8.6 Updating the operating system of the HMI device

If possible, use the interface providing the highest bandwidth for this connection, e.g. Ethernet. When you update the operating system, the runtime software on the HMI device is also updated and the installed Runtime version is changed.

NOTICE
Updating the operating system deletes all data on the HMI device When you update the operating system you delete data on the target system. For this reason, you should back up the following data beforehand: • User management • Parameter sets • Project data

Requirement

- The HMI device is connected to the configuration PC.
- The HMI device is selected in the project tree.

Updating the operating system

Proceed as follows to update the operating system:

1. Select "Online > HMI device maintenance > Update operating system" in the menu.
   The "Update operating system" dialog box opens.
2. Select the type of the PG/PC interface and the target device, and click "Update".
   The "SIMATIC ProSave [OS-Update]" dialog opens. The path to the image is preset.
3. If required, you can select a different path for the image that you want to transfer to the HMI device.
4. Click "Update OS".

This starts the update. The update operation can take time, depending on the connection selected.

Resetting the HMI device to factory settings

To reset the HMI device to factory settings, follow these steps:

1. Switch off the power supply for the HMI device.
2. Connect the HMI device to the configuration PC.
3. Select the "Update operating system" command from the menu under "Online > HMI Device maintenance" on the configuration PC in WinCC.
   The "Update operating system" dialog box opens.
4. Select the type of the PG/PC interface and the target device, and click "Update".
   The "SIMATIC ProSave [OS-Update]" dialog opens. The path to the image is preset.
5. If required, you can select a different path with a different image that you want to transfer to the HMI device.
6. Activate "Reset to factory settings".
7. Click "Update OS".
8. To reset to factory settings, switch on the power supply to the HMI device again.
   This operation can take time.

Result

The operating system of the HMI device is operational and up to date.

See also

Updating the operating system (Page 7166)

Backing up and restoring data of the HMI device (Page 7165)

Basics for downloading projects (Page 7147)

16.2.8.7 Updating the operating system of the HMI device from a data storage medium

Introduction

You can update the operating system using a data storage medium. You can find the HMI device image files, for example, in the installation directory of WinCC under: "\Siemens\Automation\Portal V1x\Data\Hmi\Transfer\<HMI device image version>\Images".

NOTICE
Data loss All data on the HMI device, including the project and HMI device password, is deleted during a restore operation. Back up data before the restore operation.

Requirement

- The HMI device image file is located in the "SIMATIC.HMI\Firmware\" directory on your data storage medium, e.g. a SIMATIC HMI Memory card or an industry-grade USB stick.
- The data storage medium with the relevant HMI devices image file including operating system is inserted in the HMI device.

Procedure

1. Open the Control Panel on the HMI device.
2. Select "Service & Commissioning > Update OS".
3. Select a storage medium under "Select storage media for OS update".

   Note

   If there is no storage medium or a defective storage medium in the HMI device, the "0 devices found" alarm is displayed. Insert the storage medium or replace the storage medium.

4. Select the required HMI device image file under "Firmware files on external storage".
5. Press "Update OS".
   The "Update OS Image" dialog opens.
6. To start restoring the operating system, press "Yes".
   The "Transfer" dialog opens. A progress bar shows the course of the restore. The HMI device then restarts.

Note

After the restore, it may be necessary to recalibrate the touch screen.

See also

Updating the operating system of the HMI device (Page 7167)

16.3 WinCC Unified PC

16.3.1 Specifying runtime settings

16.3.1.1 Introduction

Before you compile and download a project, update the runtime settings of the HMI device. You specify the Runtime languages, for example, or activate "Collaboration".

To edit the runtime settings for an HMI device, select "Runtime settings" in the project tree below the HMI device.

16.3.1.2 General

Identification

The runtime ID is the unique identification of a runtime project. The runtime ID is also stored in the SIMATIC Runtime Manager. Based on the runtime ID, you can determine whether a project has already been downloaded to the HMI device.

Note

When you rename the WinCC Runtime of the Unified PC in the Engineering System, the Runtime ID changes. The project must be downloaded completely during the next download.

Encrypted transfer

To download the project encrypted, the same password must be stored in the runtime settings and in runtime. Alternatively, transfer the password unencrypted during the initial download.

When you manage an encrypted project in the Runtime Manager, the password must be stored in the Runtime Manager.

Screen

Under "Screen", you specify the start screen and the style of the HMI device.

The start screen is displayed after runtime start.

Select the screen resolution. The setting of the HMI device is used by default.

Note

Display of a changed start screen

You have defined a start screen in the project and started runtime. When you define another start screen and download only changes to the device, the last active screen is displayed in runtime.

After reloading, refresh runtime in the browser with the <F5> key or the "Update" button.

See also

Secure communication (Page 7129)

16.3.1.3 Alarms

Controller alarms and diagnostics

Note

WinCC only supports controller alarms of a SIMATIC S7-1500. In addition, WinCC only supports controller alarms that are automatically updated by the central alarm management in the PLC.

One or more HMI connections to a PLC are shown in "Controller alarms and diagnostics".

You can manage the following options:

- "Display classes": You filter the controller alarms via display classes. You specify for each connection which display classes are displayed on the HMI device.
- "Automatic update": Controller alarms are automatically updated on the HMI device.
  Requirement: The "Central message management in the PLC" option is enabled.
- "System diagnostics": System diagnostic alarms can be displayed in runtime.
  Requirement: The "Automatic update" option is enabled.
- "Security events": Security events can be displayed in runtime.
  Requirement: The "Automatic update" option is enabled.

State texts

You specify the state texts of alarms in the runtime settings. The state texts are displayed in runtime in the alarm control.

Specify the state texts in other languages under "Languages & Resources > Project texts".

See also

Configuring state texts of alarms (Page 733)

Configuring the display of system diagnostic alarms (Page 767)

Configuring the display of security events (Page 795)

Filtering controller alarms via display classes (Page 748)

16.3.1.4 Process diagnostics

General

Indicates whether the device participates in process diagnosis.

16.3.1.5 Services

SMTP communication

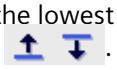
SMTP communication enables automatic sending of e-mails when events occur.

Specify the port for SMTP communication.

16.3.1.6 Language & font

Runtime language and font selection

You configure project languages that are available as runtime languages for the respective device. You also define the order in which the languages are switched.

When using central user management, runtime is displayed in the language that a user selected in the "User login" dialog during login. If this language is not configured in the runtime settings of the HMI device or if the language setting in the central user management was not set, runtime is displayed in the language that has the lowest number in the "Order" column in the runtime settings. You change the order with  .

If users do not select a language in the "User login" dialog, runtime is displayed in the language that is set for the browser.

The fixed font 1 is always provided for the respective HMI device.

When you enable the option "Enable for logging" for a language, alarm texts are logged in the respective language. To keep the size of the log relatively small, log only alarm texts in the required languages.

See also

Enabling the runtime language (Page 246)

Editing log contents with scripts and system functions (Page 850)

16.3.1.7 Collaboration

General settings

- Select the "Enable collaboration" check box.

Identification

The following information must be unique for all devices participating in a collaboration:

- System ID:
  The *system ID* must be unique for each device participating in Unified Collaboration, as this ID is used for communication between the devices.
- Collaboration name:
  Assign a collaboration name or select "Generate collaboration name automatically".
- IP address / Host name

Connect actively to

A list of all HMI devices available for collaboration with system ID and IP address / host name is displayed. You select the HMI devices to which the current HMI device provides collaboration.

Note

Collaboration is enabled in both directions

The HMI device that you select under "Connect actively to" can display screens of the current HMI device and vice versa. Collaboration is always enabled in both directions, even if you only have the connection activated for one collaboration device.

See also

Defining collaboration settings (Page 7572)

16.3.1.8 Storage system

You specify the storage locations of the following databases:

- Database for tag persistency
- Database for data logging
- Database for alarm logging

Note

Logging on network drives

Do not save databases directly on a network drive. Power supply can be interrupted at any time. Reliable operation is therefore not guaranteed.

For example, save the logs on your local hard drive or a USB stick.

If you have defined a storage location in the runtime settings of the HMI device, you can specify the database storage location for individual logs in the "Logs" editor under "Storage medium". You have the following options:

- Default:
  The database is saved according to the runtime settings of the HMI device.
- Local:
  Specify a path.

Database type

Specify the database type. WinCC Unified PC supports the following database types:

- SQLite
- Microsoft SQL
  A separate installation is required for using Microsoft SQL. After installing Microsoft SQL, SQLite logging is no longer possible.

Database storage location for tag persistency

You can specify tag persistency for internal tags. The last values of the persistent tags are used after Runtime has been started.

A separate database is used for tag persistency. The values of the tags are available again after restarting runtime or restarting or switching off the HMI device.

Specify the storage location of the database:

- Off:
  Tag persistency is not used.
- Local:
  Specify a path.
- Project folder:
  The database is stored in a subfolder of the runtime project folder.

Note

Use different storage media for tag persistency and logging.

The tag persistency can be affected if the log databases and the database for tag persistency are stored on the same storage medium and the storage medium is changed while runtime is running.

Location of the main database for data logging

Specify the storage location of the databases for data logging:

- Off:
  Tags are not logged.
- Local:
  Specify a path.
- Project folder:
  The logs are stored in the "TLGDB" subfolder of the runtime project folder.
- Default:
  In WinCC Unified Configuration you store the path at which the logs are saved under "Log settings".

Location of the main database for alarm logging

Specify the storage location of the databases for alarm logging:

- Off:
  Alarms are not logged.
- Local:
  Specify a path.
- Project folder:
  The logs are stored in the "ALGDB" subfolder of the runtime project folder.
- Default:
  In WinCC Unified Configuration you store the path at which the logs are saved under "Log settings".

See also

Specify tag persistency (Page 636)

Storage locations of logs (Page 843)

16.3.1.9 Settings for tags

You have the option of synchronizing HMI tags with the connected PLC tags. The position of a data value in the structure of a data block is thus mapped in the HMI tag name. If necessary, the PLC name is set as a prefix. You synchronize the tags in the "HMI tags" editor.

You specify before the synchronization whether the names are matched and under which conditions.

Synchronization of the name of the PLC tag in the engineering station

To avoid conflicts within complex tag types, configure how the delimiters of the path statement from STEP 7 are replaced in WinCC similar to name matching:

- Replace delimiters
  Depending on your selection, the delimiters of all hierarchy levels are replaced during synchronization.
- Replace invalid characters
- PLC prefix
  The PLC name is set as a prefix to the HMI tag name. You set the option for each HMI connection.

Note

Duplicate tag names

If the generated tag name is already in use, a number is added in parentheses, e.g. Datablock_1_Static_2{1}(1).

Example

The "PLC1" controller contains the structured data block "DB1". The "Db1.a[1].b.c[3]" data block element is used in a picture. Depending on your settings, the HMI tag name is generated as follows:

Selected option	HMI tag name
PLC prefix	Plc1.Db1.a[1].b.c[3]
Replace dot and parenthesis with ; ( )	Db1;a(1);b;c(3)
Replace dot and parenthesis with _ { } PLC prefix	Plc1_Db1_a{10}_b_c{3}

See also

Synchronizing tags (Page 640)

16.3.1.10 Good Manufacturing Practice

Traceability and therefore the documentation of production data is becoming increasingly important in many sectors such as the pharmaceutical industry, the food and beverage industry, and the related mechanical engineering industry.

Therefore, sector-specific and cross-industry standards have been developed for the electronic documentation of production data.

The most important set of regulations is the FDA guideline 21 CFR Part 11 for electronic data records and electronic signatures issued by the FDA, the US Food and Drug Administration. In addition, different EU regulations apply, such as EU 178/2002, depending on the industry.

Requirements for production systems in these industries have been developed on the basis of 21 CFR Part 11 and the corresponding layout to comply with GMP (Good Manufacturing Practice). They are also required for other industries.

The following main requirements are derived from these directives and regulations:

- Creation of an Audit Trail or operating trace in runtime
  Based on this document, it is possible to trace the user who carried out the operator action on the machine at what time.
- Important process steps must also be assigned to a clear responsibility, for example, via an electronic signature.

GMP (Good Manufacturing Practice)

1. Enable GMP-compliant configuration.
2. If necessary, select a text list entry that contains the reason for the GMP-relevant tags.

16.3.1.11 User management

You specify whether you are using a local or a central user management. By default, the use of the local user management is specified in the engineering system.

Local users are only valid for this project.

You manage central users in the TIA User Management Component (UMC).

Configuration of user management

- When you activate the local user management, you use the users and user roles that you have created under "Security settings > Users and roles" for management.
- When you activate the central user management, the users, user roles and their rights are applied from the TIA User Management Component (UMC). To access the UMC, you must specify the server address and the server ID.

See also

Local and central user management (Page 6889)

16.3.1.12 OPC UA server

General

OPC is a standardized manufacturer-independent software interface for data exchange in automation engineering. OPC UA is the technology succeeding OPC. OPC UA is platform-independent and supports different protocols as communication medium.

To work with OPC UA in WinCC Unified, the OPC UA server must be enabled in the TIA Portal in the Runtime settings of the HMI device.

Read/write tags and register tags/alarms

When you enable the "Operate as OPC UA server" option in the HMI device, the protection for unauthorized internal and external access is downgraded.

- Enable the "Operate as OPC UA server" option.
  A security note is displayed.

After enabling the option, all other settings of the OPC UA Server will become available.

Alarms and Conditions

- To display alarm conditions in the address range of the server, select the option "Enable Alarms and Conditions on the OPC UA server".
- To disable or acknowledge alarms on the OPC UA Client, for example, select the option "Allow operation of alarms on the OPC UA Client". To enable this option, the "Enable Alarms and Conditions on the OPC UA server" option must be enabled.

Options

General

Define the following settings:

- Port
  Default value: 4890
  Do not use a port number that is already assigned to another application.
- Maximum session timeout (s)
  Default value: 600000 s
- Maximum number of OPC UA sessions
  Default value: 100

Subscriptions

Define the following settings:

- Minimum publication interval (ms)
  Default value: 100 ms
- Maximum number of monitored items
  Default value: 0

Security

Secure connection

Security policies

CAUTION

Reduced security

When the option "No OPC UA Server Security" is enabled, any OPC UA client can connect to the OPC UA server regardless of the following settings.

The following section contains a list of all security policies available on the server.

- Activate the required security policies.

User authentication

Guest authentication

- To allow access by anonymous users to the OPC UA server, enable the option "Enable guest authentication".
  An authentication by means of user name and password is not required for guests.
  Security is restricted to the degree that you determine by assigning rights to this user.

Authentication by means of user name and password

- To allow access by users with user name and password to the OPC UA server, enable the option "Authentication with user name and password".
  If access to the OPC UA server is to require the user name and password, the user must be assigned the role "HMI Administrator". The "HMI Administrator" role has the system-defined "OPC UA - read and write access" function right. The settings made must then be synchronized with the user management in runtime.

16.3.1.13 Layers

Default names of the layers

Adjust the default names of the layers.

16.3.1.14 Reporting

Report system

For Unified PCs with device version V18, you make the settings for Reporting here.

"Enable Reporting"	Enables Reporting. You can create report templates based on this project or project report jobs and generate reports in Runtime only if reporting is enabled.	
"Storage location for the Reporting database"	Configure the storage location of the Reporting database. The Reporting database stores the actions and settings made in Runtime in the "Reports" control.	
	"Storage location"	"Default" The database is in the folder that is selected during installation or later in the "Reporting" step in WinCC Unified Configuration.
		"Project folder" The database is in the project folder of the Runtime project.
		"Local" The database is in the device folder you entered under "Folder".
	"Folder"	If you have selected the value "Local" under "Storage location", enter the path to the local folder here.
"Storage location for reports"	Configure the local main storage location for the generated reports. In Runtime, the local main storage location is one of the possible storage locations for reports with the "File system" target type.	
	"Storage location"	"Default" The folder defined during installation or later in WinCC Unified Configuration in "Reporting" step is used as the main local storage location.
		"Project folder" The project folder of the Runtime project is used as the main local storage location.
		"Local" The device folder you entered under "Folder" is used as the main local storage location.
	"Folder"	If you have selected the value "Local" under "Storage location", enter the path to the local folder here.

Note

Devices with a device version lower than V18

Unified PCs with a device version of lower than V18 always have the same settings:

- Reporting is always enabled.
- The folder configured during the installation of Unified Runtime or later with WinCC Unified Configuration is always used as local main storage location for reports.
- The Reporting database is always in the project folder.

16.3.2 Compiling a project

Basics

Compilation results in a runtime project that is executable on the respective HMI device.

A project can be compiled explicitly or implicitly:

- Explicit: The compilation is started manually. Compile the entire project or only changes. Explicit compiling is described in the following section.
- Implicit: The project is downloaded without previous manual compilation.
  The project is compiled in the background even as you are configuring it in WinCC.
  This reduces the compilation times before downloading the project.

Note

Changes to S7 blocks are not automatically compiled in the background. If you are using HMI tags in the project that are connected to PLC tags, you should also compile all changed S7 blocks before you compile the HMI device.

Requirement

- A project is open.

Compiling changes of a project

1. Select an HMI device in the project tree.
2. There are two ways to start compiling the changes:
   – Press the "Compile" button in the toolbar.
   – Select "Compile > Software (only changes)" from the shortcut menu.

Compiling a project completely

1. Select an HMI device in the "Project tree" area.
2. Select "Compile > Software (rebuild all)" from the shortcut menu.

Note

When a project was compiled completely, the project must also be downloaded completely afterward.

Compiling projects of multiple HMI devices simultaneously

1. Select the HMI devices using multiple selection in the project tree.
2. To compile the changes to the projects, select the "Compile" button in the toolbar. Alternatively, select "Download to device > Software (only changes)" from the shortcut menu.
3. To compile the projects completely, select "Download to device" > "Software (rebuild all)" from the shortcut menu.

Result

The configuration data of the selected HMI devices is being compiled.

If errors occur during compilation, the errors are shown in the Inspector window under "Info > Compile".

1. Correct the errors and recompile the projects.
2. Load the compiled projects.

See also

Overview (Page 7125)

Basics of downloading projects (Page 7182)

Complete reloading of a project (Page 7187)

Restrictions in compiling and loading changes (Page 7132)

16.3.3 Downloading projects

16.3.3.1 Basics of downloading projects

Introduction

During downloading, the compiled project is transferred to an HMI device. The project can be downloaded either locally to the Unified Runtime of the configuration PC or a connected HMI device. If there is no connection, an external storage medium can be used for the transfer.

Before the download, use the "Load preview" dialog to determine whether existing data is retained and whether logs are reset. The following data can be retained or overwritten:

- Runtime values: Tag values, active alarms and user data
- Logs: Data logs, alarm logs, Audit Trails and context logs

The compilation of the project is checked before downloading and missing content is compiled. This ensures that the latest version of the project is always downloaded.

When you download a project again, you can decide whether you want to download only changes or the complete project. If you want to download only changes, a compilation of changes must be possible beforehand.

Ethernet connection

You download the project to the HMI device via an Ethernet connection. The connection uses Ethernet port 20008.

Note

Ethernet port 20008

If an application already occupies Ethernet port 20008, loading is not possible.

If no connection to the target can be established, check the port assignments. If another application is using Ethernet port 20008, close this application.

Runtime ID

At the start of the configuration, each project receives a runtime ID which is transferred to the HMI device during downloading. If you have already downloaded a project, the download process recognizes the project using the Runtime ID. When you rename the WinCC Runtime of the Unified PC, the Runtime ID changes as well.

Note

Existing runtime projects on the target HMI device

If a runtime project with the same Runtime ID is already available on the HMI device, the project is overwritten.

Synchronization of IDs:

In runtime, objects and their relevant Runtime data are identified by IDs. If the Engineering data and Runtime data are not synchronized, conflicts may arise, which may lead to undesired reactions at runtime. This is the case, if, for example:

- the Engineering process is not linear, i.e. an earlier project version is enhanced with new objects and loaded to the device that is already running with a newer project version, for example, when using a backup or if the project was not saved after loading.
- the process of compiling and loading a project gets aborted unexpectedly.
- the Runtime data are reset.

To detect and rectify such application cases, the dialog "Load Preview" contains setting options for synchronizing IDs.

Note

Synchronization is not necessary when downloading changes since the linearity has already been selected.

Downloading various runtime projects onto one HMI device

If the runtime software "WinCC Unified PC RT" is installed on the PC, you can download multiple projects directly to the configuration PC. But you are only executing one runtime project in runtime. You can use the SIMATIC Runtime Manager to start and stop projects.

Example: The "Mixing" project is loaded to the configuration PC and is being executed in runtime. When you change the "Mixing" project, download the changes using "Download to device > Software (only changes)". Runtime continues running.

When you then download the "Bottling" project completely, the "Mixing" project Runtime is stopped. If you have selected the "Start runtime" option, the "Bottling" project is started.

Downloading a runtime project to multiple HMI devices

You can download a runtime project to several connected HMI devices one after the other and start runtime at the same time. Changes can only be downloaded for the first device.

See also

Complete reloading of a project (Page 7187)

Overview (Page 7125)

Compiling a project (Page 7181)

Secure communication (Page 7129)

Download changes only (Page 7190)

Loading projects of multiple HMI devices simultaneously (Page 7191)

Restrictions in compiling and loading changes (Page 7132)

Using external storage medium (Page 7192)

Compiling and loading with team engineering (Page 7194)

Basics on version compatibility (Page 187)

16.3.3.2 Initial download of a project

Introduction

The first download of a project is different from any subsequent download processes:

- The connection to the HMI device must be set up before the download.
- The project is always downloaded completely during the first download.

Note

Runtime is stopped during a complete download

The project running in runtime is stopped when a project is executed in runtime while you are completely downloading the project.

Requirement

- The project has been compiled without errors.
- A user is configured.
- The HMI device is connected to the configuration PC or the configuration PC is the HMI device.
- The Runtime version of the target device corresponds to the configured Runtime version.
- Ethernet port 20008 in the network configuration is not allocated.

Editing connection parameters before download

1. Select the HMI device in the project tree.
2. Select "Online > Extended download to device".
   The "Extended download" dialog opens.
3. Enter the IP address or device name of the new target device.
4. Click "Connect" and load the project.
   The Runtime project is downloaded with changed connection parameters.

Options when initially loading a project

Runtime values: Tag values, active alarms and user data

Select whether you want to keep the current values or reset the values to the start values.

Select "Reset to start values" at the time of initial loading.

Reset logs: Data logs, alarm logs, Audit Trails and context logs

Select whether you want to reset all logs or no log in Runtime.

Synchronizing IDs

Select whether the IDs of objects in the Engineering System and their relevant Runtime data should be synchronized.

Note

At the time of initial loading, the IDs of objects in the Engineering System and their relevant Runtime data are not synchronized, regardless of the selected option.

- Synchronize: The synchronization of IDs is verified before loading. If inconsistencies are reported during the verification of the IDs, the IDs are synchronized. Then the project is completely loaded.
- Verifying the synchronization: The synchronization of IDs is verified before loading. If inconsistencies are reported during the verification of the IDs, loading is cancelled. The IDs are then not synchronized.
- Do not Sync: Synchronization of the IDs is not verified. The system cannot guarantee that the data loaded from the Engineering System match the data present in Runtime.

  Note

  Restart Runtime

  To prevent data inconsistencies, restart Runtime when you select "Do not Sync".

Initial loading of a project

1. Select the HMI device in the project tree.
2. In the toolbar, select the "Download to device" button or select "Download to device" > "Software (all)" from the shortcut menu.

   Note

   When you select "Download to device" > "Software (only changes)" from the shortcut menu, the project is still downloaded completely during the initial download.

   The "Extended download" dialog opens.
3. Enter the address or the name of the target device. You have the following options:
   – "Configured IP address"
   – "Use other IP address"
   – "Use device name (DNS)"

   If you use your configuration PC as an HMI device, enter the IP address 127.0.0.1 or the device name "localhost".
4. Click "Connect".
   The connection is established and a dialog is displayed.
5. Select "Load".
   The compilation of the project is checked and missing content is compiled.
   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.

6. Check the displayed default settings and change the settings as necessary:
   - Select the "Full download" option under "Load Runtime".
   - Under "Runtime start", specify whether Runtime should start on the target system after the download.
   - Under "Runtime values", select "Reset to start values" in the Options menu.
   - Under "Reset logs", specify whether all logs are reset in Runtime.
     The setting is only accepted when you have selected "Start runtime".
7. Click "Download".

Result

The project is downloaded onto the selected HMI device under the file path "C:\ProgramData\SCADAProjects".

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

On completion of the successful download of the project, you can execute it on the HMI device. If you have activated the start of runtime on the target system in the "Load Preview" dialog, the project is started in runtime after loading.

See also

Specify tag persistency (Page 636)

Basics of downloading projects (Page 7182)

Compiling a project (Page 7181)

Secure communication (Page 7129)

Overview (Page 7125)

Loading projects of multiple HMI devices simultaneously (Page 7191)

16.3.3.3 Complete reloading of a project

Options when reloading a project

Make the following settings in the "Load preview" dialog during reloading:

- Load Runtime: Full download
  You cannot download if you keep the "No action" option.
- Runtime start: Specify whether Runtime should start on the target system after the download.

- Runtime values: Select whether you want to keep the current values for tags, active alarms and user administration data or if you want to reset the values to the start values.
  With the "Keep selected" option, you can specify the values you want to keep.
  - Current values of tags and pending alarms
  - Current user administration data: Disable "Keep current user administration data in runtime" during initial download.
- Reset logs: Select whether you want to reset data logs, alarm logs, Audit Trails and context logs or no log in Runtime.
- Synchronize IDs: Specifies the process to be followed for synchronizing the IDs of objects in the Engineering System and their relevant Runtime data.
  The following options are available:
  - Synchronize: The synchronization of IDs is verified before loading. If inconsistencies are reported during the verification of the IDs, the IDs are synchronized. Then the project is completely loaded.
  - Verifying the synchronization: The synchronization of IDs is verified before loading. If inconsistencies are reported during the verification of the IDs, loading is cancelled. The IDs are then not synchronized.
  - Do not Sync: Synchronization of the IDs is not verified. The system cannot guarantee that the data loaded from the Engineering System match the data present in Runtime.

  Note

  Restart Runtime

  To prevent data inconsistencies, restart Runtime when you select "Do not Sync".

Requirement

- The project has been compiled without errors.
- The project has been downloaded at least once before.

Procedure

1. Select the HMI device in the project tree.
2. Select "Download to device > Software (all)" from the shortcut menu.
   The compilation of the project is checked and content that has not been compiled is compiled.
   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.

   Note

   Runtime is stopped during a complete download

   The project running in runtime is stopped when a project is executed in runtime while you are completely downloading the project.

3. Check the displayed default settings and change the settings as necessary:
   - Select the "Full download" option under "Load Runtime".
   - Specify whether Runtime should start on the target system after the download.
   - Specify whether tag values, active alarms, and user data are retained.
     The setting is only accepted when you have selected "Start runtime".
     To retain internal tags, the persistency must be enabled in the settings of the respective tag.

     Note

     To prevent the users created in the user administration from being overwritten in runtime by the complete download of the project, activate the "Keep current user administration data in runtime" option.

     When this option is selected, role assignments and function rights from the user administration of the Engineering System are loaded to Runtime, but not user data such as user name and password.

   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".
   - Select the desired option under "Synchronize IDs" in the selection menu.
4. Click "Download".

Result

The project is downloaded onto the selected HMI device under the file path "C:\ProgramData\SCADAProjects".

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

After the successful download of the runtime project, you can execute it. If you have activated the start of runtime on the target system in the "Load Preview" dialog, the project is started in runtime after loading.

See also

Basics of downloading projects (Page 7182)

Overview (Page 7125)

Compiling a project (Page 7181)

Specify tag persistency (Page 636)

Secure communication (Page 7129)

16.3.3.4 Download changes only

Introduction

When you only download changes to a project, the relevant project must be executed in runtime. Runtime is not closed when loading.

Requirement

- The project has been compiled without errors.
- The project has been downloaded at least once before.
- A compilation of changes must be possible or have been executed.
- The project that contains the changes is being executed in runtime.

Procedure

1. Select the HMI device in the project tree.
2. Press the "Download to device" button in the toolbar.
   Alternatively, select "Download to device > Software (only changes)" from the shortcut menu.
   The compilation of the project is checked and content that has not been compiled is compiled.
   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.
3. Click "Download".

Result

The changes are downloaded onto the selected HMI device under the file path "C:\ProgramData\SCADAProjects".

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

Runtime continues to be executed.

Update Runtime in the browser with the <F5> key or by pressing the "Update" button to see the changes in runtime.

See also

Basics of downloading projects (Page 7182)

Restrictions in compiling and loading changes (Page 7132)

16.3.3.5 Loading projects of multiple HMI devices simultaneously

Requirement

- Several WinCC Unified PCs are configured.
- The individual projects have been compiled without errors.
- A user is configured.
- Ethernet port 20008 in the network configuration is not allocated.

Connecting WinCC Unified PCs to the configuration PC

1. Select a WinCC Unified PC in the project tree.
2. Select "Online > Extended download to device".
   The "Extended download" dialog opens.
3. Enter the address or the name of the target device. You have the following options:
   - "Configured IP address"
   - "Use other IP address"
   - "Use device name (DNS)"
4. Click "Connect".
   The connection is established and a dialog is displayed.
5. To connect additional HMI devices and load several projects at the same time, select "Cancel".
6. Repeat steps 1 to 5 for additional WinCC Unified PCs.

Loading projects

1. Select the WinCC Unified PCs using multiple selection in the project tree.
2. To download the changes to the projects, select the "Load" button in the toolbar.
   Alternatively, select "Download to device > Software (only changes)" from the shortcut menu.
3. To download the projects completely, select "Download to device" > "Software (all)" from the shortcut menu.
   The compilation of the projects is checked and content that has not been compiled is compiled.
   The result of the compilations is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed. All selected projects are listed in the dialog.

   Note

   Runtime is stopped during a complete download

   The project running in runtime is stopped when a project is executed in runtime while you are completely downloading the project.

4. Check the displayed default settings and adjust the settings for each device:
   - Specify whether runtime should start on the target system after the download.
   - Specify whether tag values, active alarms, and user data are retained.
     The setting is only accepted when you have selected "Start runtime".
   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".
5. Click "Download".

See also

Initial download of a project (Page 7185)

Basics of downloading projects (Page 7182)

16.3.3.6 Using external storage medium

Loading project to external storage medium

Introduction

If you cannot establish a direct connection from the configuration PC to the HMI device, load the compiled runtime project onto an external storage medium. For example, use a USB flash drive or SD card.

You load either the complete runtime project or only changes of a runtime project.

As soon as you have connected the external storage medium to your HMI device, transfer the project to the HMI device.

Requirement

- An HMI device has been created.
- The project has been compiled without errors.
- A user is configured.

Procedure

1. Jump to the "Devices" tab in the project tree.
2. Double-click "Add user-defined card reader" in the "Card reader/USB storage" folder.
   A selection dialog opens.
3. Select a target directory to save the project.

4. Drag and drop the HMI device (e.g. "HMI_1 [<Device type>]") to the added folder.
   Alternatively, use copy and paste.
   The project is checked. If the project has contents that have not yet been compiled, a compile is performed.
   The "Load Preview" dialog opens.
5. In the selection menu you specify which project contents are downloaded:
   - "Full download": The project is downloaded completely.
   - "Delta download": Only changes of the project are downloaded.
6. Click "Load" to confirm.

Result

Your project is stored as a compressed ZIP folder in the directory "[<Target directory>]\Simatic.HMI\RT_Projects". The file name is made up of the name of the HMI device, the project name and the time stamp:

- Projects that were created with the option "Full download" receive as file name e.g. "HMI_RT_1[Project1] - Full 2020-04-03 - 14.51.41.zip".
- Projects that were created with the option "Delta download" receive as file name e.g. "HMI_RT_1[Project1] - Delta 2020-04-03 - 14.53.45.zip".

If errors or warnings occur during the download, corresponding alarms are output under "Info > General" in the Inspector window.

See also

Basics of downloading projects (Page 7182)

Load project from external storage medium

Use the SIMATIC Runtime Manager to download a project from an external storage medium.

See also

Adding a project (Page 7497)

16.3.4 Compiling and loading with team engineering

16.3.4.1 Basics on compiling and loading with team engineering

Introduction

You can compile and download to an HMI device in the server project view, in the exclusive session and the local session.

Note

Unified objects cannot be selected in a local session

To edit an object using Multiuser Engineering, it must first be "selected". Only objects marked for check-in can be transferred into the server project after editing.

Unified objects cannot be marked in a local session. Changes to these objects are not applied to the server project during check-in.

You can edit unmarked objects in the server project view.

Basics

The options for compiling and downloading in the server project view, in the exclusive session or the local session are no different from the options in a single-user project. The most recent project is always compiled or loaded from the currently active view.

You can execute all commands for compiling and downloading in multiuser engineering and exclusive engineering:

- "Software (rebuild all)"
- "Compile > Software (only changes)"
- "Software (all)"
- "Download to device > Software (only changes)"

Rules

The following rules apply to compiling and downloading in multiuser engineering and exclusive engineering:

- The project that was changed in a local session always remains local and is not uploaded to the multiuser server.

  Note

  A Unified project that was created or changed in a local session cannot be saved in the multiuser server project.

  Use the local session to test your configuration. When you update your local session, all changes to Unified objects are overwritten by the server project.

- Only projects that were created or changed in the server project view or in the exclusive session can be saved in the multiuser server project.

See also

Compiling in the server project view and in the exclusive session (Page 7195)

Downloading projects (Page 7182)

YouTube (https://www.youtube.com/watch?v=n4oTZ2Gzg6U)

16.3.4.2 Compiling in the server project view and in the exclusive session

Basics

Compiling and downloading of projects in the server project view and the exclusive session is no different from compiling and downloading in a single-user project.

While you are compiling a project in the server project view or the exclusive session, the server project is blocked. Other users cannot edit the server project in the meantime. The compiled runtime project is saved with the WinCC project on the central server. Blocking the server project ensures that the configuration data and the runtime project remain synchronized.

Note

When you compile and save in the server project view or in the exclusive session, other users then obtain the Runtime project you have updated along with the WinCC project when they "refresh" their local session. Other users do not have to recompile the changes you have made after an update.

See also

Basics on compiling and loading with team engineering (Page 7194)

16.3.5 Error messages during loading of projects

Possible problems during the download

When a project is being downloaded to the HMI device, status messages regarding the download progress are displayed in the output window.

Problems arising during the download of the project to the HMI device are usually caused by one of the following errors:

- Wrong version of operating system on the HMI device
- Incorrect settings for downloading to the HMI device
- Incorrect HMI device type in the project
- The HMI device is not connected to the configuration PC.

The most common download failures and possible causes and remedies are listed below.

The download is canceled due to a compatibility conflict

Possible cause	Remedy
The configuration PC is connected to a wrong device, e.g. a PLC.	Check the cabling. Correct the communication parameters.
The database of the configured HMI device in the engineering system differs from the database type in runtime.	Runtime of the Unified PC uses Microsoft SQL. Change the database type in the runtime settings of the HMI device under "Storage System" to "Microsoft SQL".

Project download fails

Possible cause	Remedy
The connection to the HMI device cannot be established.	Check the physical connection between the configuration PC and the HMI device.

See also

Downloading projects (Page 7182)

16.3.6 Starting and stopping runtime

Introduction

You have two options for starting a project Runtime:

- Select the "Start runtime" option in the "Load preview" dialog before you download a runtime project.
  Runtime is started automatically after downloading the Runtime project.
- Use the "SIMATIC Runtime Manager".

Requirements

- The runtime project is downloaded to the HMI device.

Starting runtime

To start the Runtime of the downloaded Runtime project with the SIMATIC Runtime Manager, follow these steps:

1. Start the "SIMATIC Runtime Manager" tool.
2. Click on the project in the project list.
3. Click the "Start" button.
4. Define whether project data is reset on startup.
   - To start the project in a state that existed when the project was first started, activate the options "Reset login data" or "Reset Runtime data".
   - Disable these options to start the project in a state that existed before the last project stop.
5. Select "Start".

Stop runtime

You have two options for stopping a project Runtime:

- Select the "Online > Stop runtime/simulation" button in the Engineering System.
- Select the "Stop" button in the "SIMATIC Runtime Manager".

See also

Downloading projects (Page 7182)

16.3.7 Managing users in Runtime

Requirement

- An administrator has been created.
- The IP address or the fully qualified name (name and domain) of the PC on which Runtime is installed is entered in the browser.
  If Runtime is installed on the same PC as the browser, the "localhost" designation can also be used.

Procedure

1. Select "User management".
2. Log on as an administrator.
3. Expand the selection menu at the top right.

4. Select "Users".
5. You have the following options:
   - Create new users.
   - Change the properties of the users.

Changing your password

1. Expand the selection menu at the top right.
2. Select "User profile".
3. Enter your current password.
4. Assign a new password.
5. Enter the new password again.
6. Select "Change". The password is changed.

16.4 Simulating control with PLCSIM

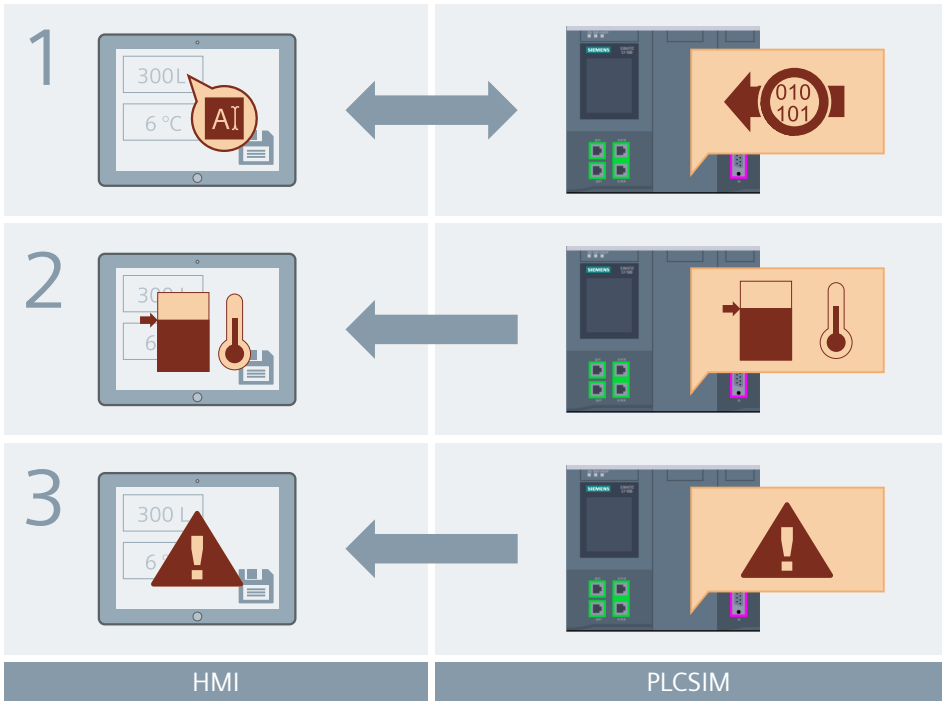
16.4.1 Using PLCSIM

Introduction

With PLCSIM you can test the configuration of your HMI device without the PLC required for it. PLCSIM simulates a PLC to which you connect the HMI device. Process parameters can be changed, or specific hardware behavior can be triggered to test the reaction of the HMI device.

With PLCSIM, you can test the following functions of your HMI device:

1. Communication between HMI device and PLC
   - Read
   - Write
2. Process behavior
   - Changing PLC parameters
   - Sequential change of recorded changes to PLC parameters
3. Hardware behavior
   - Hardware alarm
   - Diagnostic error alarm
   - Pulling or plugging modules
   - Rack/station failure



Requirements for using PLCSIM

Requirements in the software

PLCSIM is installed during the installation of WinCC. This requires you to select the corresponding check box during installation.

Requirements in the project

No special preparations are necessary to use PLCSIM. You complete their configuration in the usual way. When you start the simulation, the communication runs with PLCSIM instead of a real CPU.

16.4.2 Starting simulation and simulating behavior

Requirement

The configuration is completed.

Procedure

To test the configuration in runtime, follow these steps:

1. Select the PLC in the project tree and select "Online > Start simulation" in the menu.
   PLCSIM starts and the "Extended download to device" dialog is displayed.
2. Start the search with "Start search".
   The simulated PLC is displayed as target device.
3. Select the simulated PLC and select the "Download" command.
   PLCSIM is now ready for operation.
4. Select the HMI device in the project tree and select "Online > Start simulation" in the menu.
   The "Load Preview" dialog is displayed.
5. Select the desired settings and load the project into the HMI device.
6. Start runtime.
7. Use the editors of the project view of PLCSIM to test the behavior of the HMI.

16.4.3 Preparing simulation with PLCSIM

Requirement

- The engineering project is open.
- PLCSIM has started.

Procedure

To prepare a project with PLCSIM, follow these steps:

1. Switch from the compact view to the project view of PLCSIM by clicking on the button in the upper right-hand corner.
2. Open the "New" command via the "Project" menu and create a new project via the dialog.
3. Load and organize the PLC tags from the project in the SIM tables.
   You can load all PLC tags from an active project using the "Load project tags" button.
4. If desired, create sequences by recording the parameter changes in the SIM tables or create events in the event table.
5. Save the project.

Result

The PLCSIM project is prepared and can be used.

16.4.4 Working with PLCSIM

User interface of PLCSIM

With PLCSIM, you can choose between two different user interfaces: Compact view and project view. The view you select depends on how you want to use PLCSIM in combination with TIA Portal.

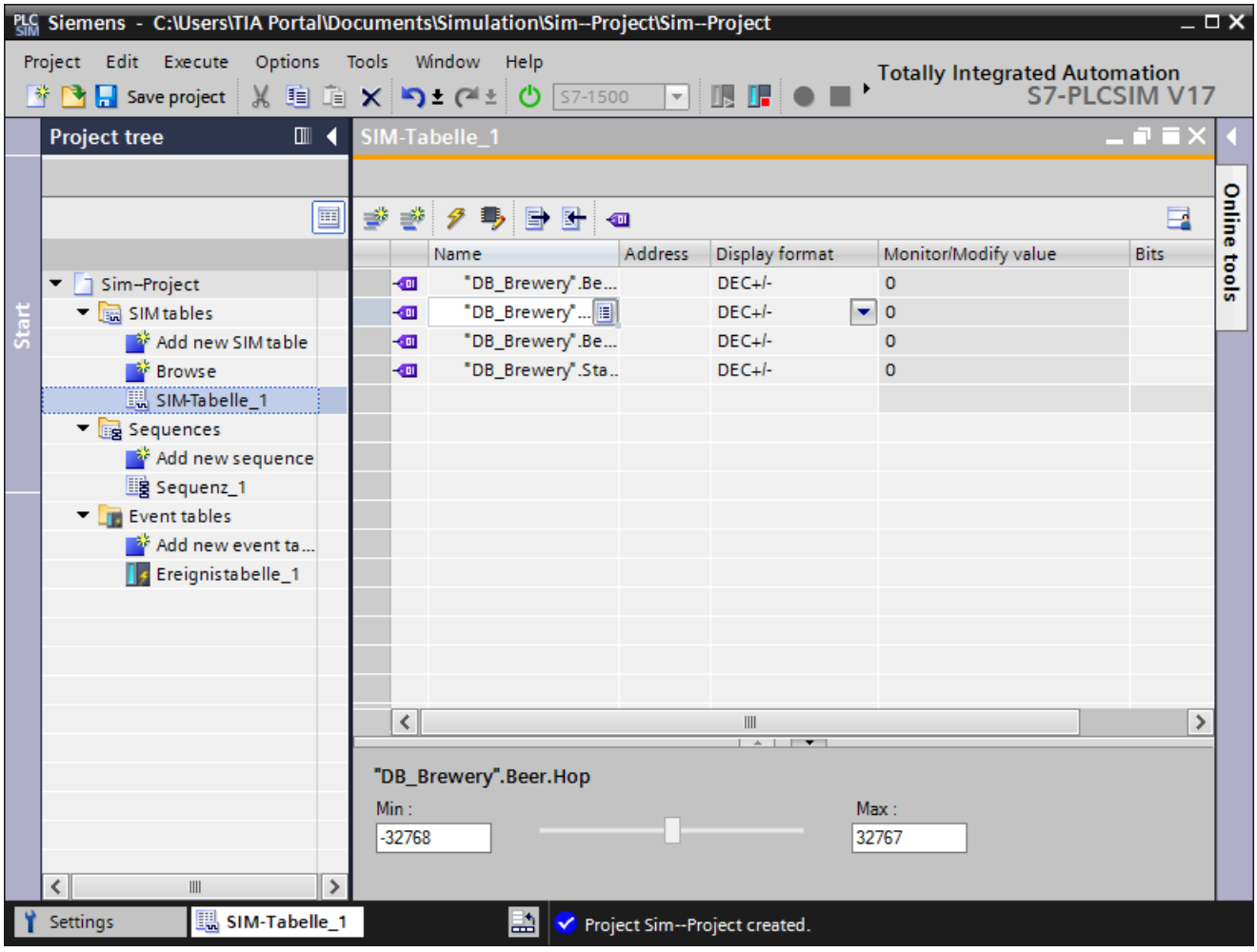
Compact view - Operate PLC

You operate the CPU exclusively in the compact view. The basic operating functions of a CPU are available with the operator controls of the compact view: "RUN", "STOP", "PAUSE" and "MRES". The status displays inform you about the state of the CPU.

Project view - Simulate process behavior and hardware behavior

The project view offers the same functions as the compact view. Additionally, simulation projects are created and stored in the project view. In the simulation project, you define the CPU behavior that you want to simulate. Several editors are available for this:

- You import PLC parameters from the automation project in Sim tables. The parameters can be written and read there.
- You simulate the behavior of external processes in sequences. To do this, you start a recording and record parameter changes that you make in the SIM table. You can edit the recorded sequence afterwards and test your process with it.
- In result tables, you select from a list of hardware actions that you want to simulate. Triggerable events are alarms, pulling and plugging of modules as well as the failure of modules or stations.



Further information

You can find more information on this topic in the PLCSIM user documentation.

Runtime and simulation 17

17.1 Simulate runtime

17.1.1 Simulate Unified Comfort Panel

17.1.1.1 Basics of simulation

Introduction

You can use the simulator to test the performance of your runtime project on the configuration PC. This allows you to quickly locate any logical configuration errors before productive operation.

The simulation is compiled like a real project and loaded into the Runtime installed on the configuration PC. To shorten the process, you can hide the "Load preview" dialog for a simulation.

The project is downloaded as follows:

- The project is fully downloaded if the simulation of the project is not executed in runtime.
- If the simulation of the project is executed in runtime and changes can be compiled and downloaded, only changes are downloaded.

You recognize a simulation in the SIMATIC Runtime Manager by the "Simulation" type. A runtime project can be downloaded to the PC as a real project and as a simulation at the same time.

With an installed Runtime as of version V17, backward compatibility is also supported for simulations.

You can simulate runtime projects with a configured Runtime version of V16 or V17.

Field of application

You can use the simulator to test the following functions, for example:

- Screen change and screen navigation
- Internal tags
- Screen display
- Configured alarms which are not triggered by a PLC

See also

Simulating a project (Page 7207)

Restrictions in compiling and loading changes (Page 7132)

Using PLCSIM (Page 7199)

17.1.1.2 Skip "Load preview" dialog

Skip "Load preview" dialog

To permanently skip the "Load preview" dialog when simulating projects, follow these steps:

1. Open the settings under "Options > Settings".
2. Select "Simulation".
3. In the "HMI Simulation" area, clear the check box "Show 'Load preview' dialog during download to simulation".

Note

Settings of the "Load preview" dialog

The following settings are applied from the previous loading process with displayed "Load preview" dialog:

- Settings for keeping tag values, active alarms and user data (default value: enabled).
- Settings for resetting logs (default value: "No reset")

If the "Load preview" dialog was hidden before the first loading of the project, the default values are used.

If other settings are required, the "Load preview" dialog can be opened via the "Online > Download to device" command.

Result

- The "Load preview" dialog is no longer displayed.
- The simulation is downloaded directly.
- The simulation is opened automatically in the standard browser.

Download messages that occur are displayed in the Inspector window in the "Info > Load" tab.

See also

Simulating a project (Page 7207)

17.1.1.3 Simulating a project

Introduction

You simulate a project on the configuration PC and download the simulation via an Ethernet connection. The connection uses Ethernet port 20008.

Note

Ethernet port 20008

If an application already occupies Ethernet port 20008, loading is not possible.

If no connection to the target can be established, check the port assignments. If another application is using Ethernet port 20008, close this application.

Requirement

- The "SIMATIC WinCC Unified Runtime" component is installed on the configuration PC.
- The project is open in the configuration PC.
- The PLC and the HMI device have been compiled successfully.
- A user is configured.

Procedure

To start the simulation with enabled "Load preview" dialog, follow these steps:

1. Select one of the following buttons:
   - From the shortcut menu of the HMI device: "Start simulation"
   - In the "Start simulation" toolbar
   - In the "Online > Simulation > Start" menu
   - In the portal view "Visualization > Simulate device".

   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.

NOTICE
Running runtime is stopped during complete download of a simulation The project running in runtime is stopped when a project is executed in runtime on the configuration PC and a simulation is downloaded completely.

2. When the simulation is downloaded completely, check the displayed default settings and change the settings as necessary:
   - Specify whether runtime should start after the download.
   - Specify whether tag values, active alarms, and user data are retained.
     Only available if you have selected "Start runtime".
     To retain internal tags, the persistency must be enabled in the settings of the respective tag.
   - Specify whether all logs are reset in runtime.
     Only available if you have selected "Start runtime".
3. Click "Download".
   If you have selected the "Start runtime" option, Runtime is started after the download.
4. Open the browser.
5. Call the URL "https://localhost" in the browser.
   Instead of the name "localhost", you can use the PC name.
6. Select "WinCC Unified RT".
7. Enter the user name and password.
   The configured screen is displayed as start screen in the browser.
8. Test, for example:
   - Screen change and screen navigation
   - Layout
   - Internal tags
9. To stop the simulation, you have these options:
   - Select "Online > Stop runtime/simulation" in the menu bar.
   - Select "Stop runtime/simulation" in the shortcut menu of the HMI device.

   While the simulation is running, the project is always loaded in simulation mode.

Note

Simulation and license

If no license is found for Runtime, an alarm appears and the simulation runs in demo mode. After 1 hour, you will be reminded that no license was found.

See also

Basics of simulation (Page 7205)

Managing users in Runtime (Page 7198)

17.1.1.4 Simulating a central user management

You want to simulate a project in which a central user management is configured for a customer. You have two options if you do not have access to the central user management of the customer:

- Configure your own central user management.
- Configure a local user management.

Requirement

- You know which groups and their function rights are contained in the central user management of the customer.

Configuring a central user management

Configure a central user management for the simulation project.

1. Create the users.
2. Create the user groups according to the customer project.
3. Assign the users to the groups.
4. Establish the connection to the central user management.
5. Start the simulation.
6. Log on in runtime.

Changes can be downloaded.

Configuring a local user management

Configure a local user management.

1. Create one or more users.
2. Assign the roles to the users.
3. Start the simulation.

Changes cannot be downloaded.

17.1.2 Simulating Unified PC

17.1.2.1 Basics of simulation

Introduction

You can use the simulator to test the performance of your runtime project on the configuration PC. This allows you to quickly locate any logical configuration errors before productive operation.

The simulation is compiled and downloaded just like a real project. To shorten the process, you can hide the "Load preview" dialog for a simulation.

The project is downloaded as follows:

- The project is fully downloaded if the simulation of the project is not executed in runtime.
- If the simulation of the project is executed in runtime and changes can be compiled and downloaded, only changes are downloaded.

You recognize a simulation in the SIMATIC Runtime Manager by the "Simulation" type. A runtime project can be downloaded to the PC as a real project and as a simulation at the same time. You cannot run the real project and the simulation at the same time in runtime.

With an installed Runtime as of version V17, backward compatibility is also supported for simulations.

You can simulate runtime projects with a configured Runtime version of V16 or V17.

Field of application

You can use the simulator to test the following functions, for example:

- Screen change and screen navigation
- Internal tags
- Layout
- Configured alarms which are not triggered by a PLC

See also

Simulating a project (Page 7211)

Restrictions in compiling and loading changes (Page 7132)

Using PLCSIM (Page 7199)

17.1.2.2 Skip "Load preview" dialog

Skip "Load preview" dialog

To permanently skip the "Load preview" dialog when simulating projects, follow these steps:

1. Open the settings under "Options > Settings".
2. Select "Simulation".
3. In the "HMI Simulation" area, clear the check box "Show 'Load preview' dialog during download to simulation".

Note

Settings of the "Load preview" dialog

The following settings are applied from the previous loading process with displayed "Load preview" dialog:

- Settings for keeping tag values, active alarms and user data (default value: enabled).
- Settings for resetting logs (default value: "No reset")

If the "Load preview" dialog was hidden before the first loading of the project, the default values are used.

If other settings are required, the "Load preview" dialog can be opened via the "Online > Download to device" command.

Result

- The "Load preview" dialog is no longer displayed.
- The simulation is downloaded directly.
- The simulation is opened automatically in the standard browser.

The Inspector window displays any download messages that occur in the "Info > Load" tab.

See also

Simulating a project (Page 7211)

17.1.2.3 Simulating a project

Requirement

- The "SIMATIC WinCC Unified Runtime" component is installed on the configuration PC.
- The project is open in the configuration PC.
- The PLC and the HMI device have been compiled successfully.
- A user is configured.

Always start the simulation in WinCC and then S7-PLCSIM.

If you have not installed the TIA Portal or S7-PLCSIM yourself, you must be a member in the following Windows user groups:

- PlcSimUsers
- RTIL Tracing Users
- Siemens TIA Engineer
- SIMATIC HMI
- SIMATIC HMI VIEWER

Procedure

To start the simulation with enabled "Load preview" dialog, follow these steps:

1. Select one of the following buttons:
   - From the shortcut menu of the HMI device: "Start simulation"
   - In the toolbar: "Start simulation"
   - Menu command "Online > Simulation > Start"
   - Under "Visualization > Simulate device" in the portal view.

   The compilation result is displayed in the Inspector window under "Info > Compile".
   The "Load Preview" dialog is displayed.

NOTICE
Running runtime is stopped during complete download of a simulation A project running in runtime is stopped when a simulation is completely loaded.

2. When the simulation is downloaded completely, check the displayed default settings and change the settings as necessary:
   - Specify whether runtime should start after the download.
   - Specify whether tag values, active alarms, and user data are retained.
     The setting is only accepted when you have selected "Start runtime".
     To retain internal tags, the persistency must be enabled in the settings of the respective tag.
   - Specify whether all logs are reset in runtime.
     The setting is only accepted when you have selected "Start runtime".
3. Click "Download".
   If you have selected the "Start runtime" option, Runtime is started after the download.
4. Open the browser.
5. Call the URL "https://localhost" in the browser.
   Instead of the name "localhost", you can use the PC name.
6. Select "WinCC Unified RT".
7. Enter the user name and password.
   The configured screen is displayed as start screen in the browser.

8. Test, for example:
   - Screen change and screen navigation
   - Layout
   - Internal tags
9. You have several options for stopping the simulation:
   - Select "Online > Stop runtime/simulation" in the menu bar.
   - Select "Stop runtime/simulation" in the shortcut menu of the HMI device.

   As long as the simulation is running, the project is always loaded in simulation mode.

Note

Simulation and license

If no license is found for Runtime, an alarm appears and the simulation runs in demo mode. After 1 hour, you will be reminded that no license was found.

See also

Basics of simulation (Page 7210)

Skip "Load preview" dialog (Page 7211)

Specify tag persistency (Page 636)

17.1.2.4 Simulating a central user management

You want to simulate a project in which a central user management is configured for a customer. You have two options if you do not have access to the central user management of the customer:

- Configure your own central user management.
- Configure a local user management.

Requirement

- You know which groups and their function rights are contained in the central user management of the customer.

Configuring a central user management

Configure a central user management for the simulation project.

1. Create the users.
2. Create the user groups according to the customer project.
3. Assign the users to the groups.
4. Establish the connection to the central user management.

5. Start the simulation.
6. Log on in runtime.

Changes can be downloaded.

Configuring a local user management

Configure a local user management.

1. Create one or more users.
2. Assign the roles to the users.
3. Start the simulation.

Changes cannot be downloaded.

See also

Specifying local or central user management (Page 6894)

Managing local users (Page 6900)

17.2 Operating Unified Panel

17.2.1 Users in runtime

Introduction

To protect your project, create users and assign roles to them. You can create local users or central users in the UMC (User Management Component). To access the runtime of a project, a user must be configured before loading. You configure the user administration in the engineering system.

Local user administration

To transfer the local user administration from the engineering system to the HMI device, the HMI device "Keep current user administration data in the runtime" must be disabled in the "Load preview" dialog before loading.

Runtime starts after loading the local user administration settings.

Central user administration in the UMC

The connection to the central UMC server is established after loading the settings of the central user administration. Runtime starts and you log on in Runtime.

See also

Using central user management in the Control Panel (Page 6940)

Protecting the Control Panel from being accessed (Page 6926)

17.2.2 Viewing memory card data

17.2.2.1 Basics

WinCC provides you with the possibility of viewing data stored on your memory card. The function supports the use of memory cards of the HMI device and of the PLC.

You have the following options:

- Viewing a backup (Page 7215)
- Renaming and deleting backups (Page 7217)

See also

Viewing a backup (Page 7215)

Renaming and deleting backups (Page 7217)

17.2.2.2 Working with backups

Viewing a backup

Introduction

If you have stored the backup of an HMI device on a memory card, this backup can also be viewed in the TIA Portal.

Requirements

- WinCC is installed.
- A memory card with a backup is available.
- The card reader is connected to the configuration PC.
- The project view is open.

Backup on the memory card in the card reader

1. Insert the memory card into the card reader.
2. Open "Card Reader/USB storage" in the project tree.

3. Select the card reader drive.
   The "Online card data" folder is displayed.
4. Open the "Online card data" folder.
5. Click the backup to open the shortcut menu.
6. Select "Properties".

Backup on the memory card of the PLC

Proceed as follows if the backup is stored on the memory card of the PLC:

1. Connect the PLC with the configuration PC.
2. Click on the PLC in the project navigation.
3. Select "Connect online" from the shortcut menu.
   A connection to the PLC is established.
   Once the PLC is connected, the "Online card data" folder is displayed.
4. Open the "Online card data" folder.

   Note

   Accessing a password-protected PLC

   When you attempt to access a PLC that is protected by a password, you will be prompted to enter the password.

   You need at least read access rights in order to view the data that is stored on the memory card.

5. Click the backup to open the shortcut menu.
6. Select "Properties".

Result

The backup properties are displayed in a separate dialog:

- General properties
  - Date and time when the backup was created
  - Software version with which the backup was created.
- Supported HMI devices with which the backup is compatible

See also

Renaming and deleting backups (Page 7217)

Renaming and deleting backups

Introduction

You can rename and delete backups from a memory card in the project navigation of the TIA Portal.

Requirements

- WinCC is installed.
- The card reader is connected to the configuration PC.
  Or The PLC is connected online with the configuration PC.
- A memory card with a backup is available.
- The project view is open.
- The backup is displayed in the project navigation.

Note

Accessing a password-protected PLC

When you attempt to access a PLC that is protected by a password, you will be prompted to enter the password.

You need write access rights to rename or delete memory card data.

Procedure

1. Click on the backup in the project navigation.
2. Open the shortcut menu.
3. Select "Rename" to rename the file.
4. Enter a new name.
5. Select "Delete" to delete the file.

Result

The backup file is now renamed or deleted.

See also

Viewing a backup (Page 7215)

17.2.3 Operation in Unified Runtime

17.2.3.1 Overview

Operating options for an HMI device

The following operating options are available:

- Operation via touch screen
  The HMI device has a touch-sensitive touch screen. Use your finger or a suitable touch pen to operate the touch screen.
- Operation via mouse and keyboard
  You can use the mouse and keyboard to operate the device like a PC.

Adhere to the instructions for operating the device in the relevant operating instructions.

Individually configured operation

The configuration engineer has various options available for setting up operation.

Examples of actions whose execution is always determined on a project-specific basis:

- Screen change
- Reporting
- Changing runtime language

There are no specific operating elements to execute certain functions. The configuration engineer specifies the project-specific execution. The screen change can be triggered, for example, via a button.

Information on project-specific operations can be found in the system documentation.

17.2.3.2 Operation with the touch screen

Overview of operation with the touch screen

You use the touchscreen to operate the HMI device or the project running on your HMI device.

Special features when operating using the touch screen

Operation with the touch screen is characterized by the following special features:

- Enable
  To enable the operator control, use your finger or a suitable touch pen to operate the touch screen. To generate a double-click, touch the operator control twice in rapid succession.
- Value input
  You enter numbers and letters on the touch screen with a screen keyboard.

Input using the screen keyboard

The screen keyboard is displayed when you select a screen item that requires input. The screen keyboard is hidden again when input is complete.

Further information on the screen keyboard can be found in the operating instructions of the HMI device.

Placing the focus on objects

You have the following options:

- Click or tap on the object.

  Note

  Giving focus to objects with a transparent background

  If an object has a transparent background, click on a visible area of the object.

- Press <Tab> until the object has the focus.

Operating objects with transparent fill

The objects displayed on a screen can have transparent ranges.

Example: Sliders, bars and pointer instruments are enclosed by a transparent rectangle.

Requirement

An event which is triggered by operating actions such as typing or clicking has been configured for the object in the engineering.

Trigger event

To trigger the event, proceed as follows:

- If the object does not have the focus, click a visible part of the object, e.g. its border.
- If the object already has the focus, the event is also triggered by clicking in the transparent area.

Using multi-touch functions

Supported gestures

Definition

Various touch gestures are available for the runtime operation. Some touch gestures have different effects in the process pictures than in the controls.

Note

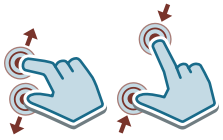
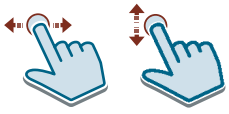
No operation with three or more fingers.

Only use one or two fingers when operating with touch gestures.

If you use more than two fingers with touch gestures, this can cause incorrect operation.

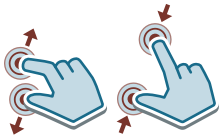
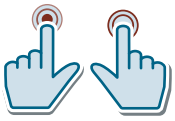
In the case of multitouch operation with several fingers, you only operate the respectively configured objects.

Supported touch gestures in process pictures

Icon	Gesture	Function
	Tap	To select an object, tip on the corresponding position in the process screen.
	Drag with one finger	To move objects with a window, drag the object by its title bar in the desired direction.
	Scale	To zoom in or zoom out, drag simultaneously with two fingers.
	Wiping	To switch between two process screens, swipe horizontally or vertically with one finger. A touch area must be configured for this function.
	Keep pressed	The function corresponds to a right-click. To trigger the event configured for the right-click, press for longer than a second on the object or the link.

Supported touch gestures in controls

Icon	Gesture	Behavior	Supported WinCC controls
	Tap	• To select a row, tap the row. • With corresponding configuration of the control: To select a cell. • To sort a column, click on the column header. • In trend controls: Zooms in on the trend area along the X/Y axis. Requirement: The "Zoom +/-", "Zoom time axis +/-" or "Zoom value axis +/-" button is pressed.	• Alarm control • Process control • Trend control • Function trend control • Ruler window • System diagnostics control • Parameter set control
	Tap with two fingers	Zooms out of the trend control. Requirement: The "Zoom +/-", "Zoom time axis +/-" or "Zoom value axis +/-" button is pressed.	• Trend control • Function trend control
	Drag with two fingers	To move window contents, such as zoomed-in tables or trends, drag with two fingers in the operating element window.	• Trend control • Process control • Ruler window • Browser
	Drag with one finger	• Moves the ruler. • Moves the X axis or Y axis. Requirement: The "Move trend area" or "Move axes area" button is pressed or the control is zoomed in.	• Trend control • Function trend control
		To select multiple rows, tap a row and drag your finger up or down. With corresponding configuration of the control: To select multiple cells.	• Alarm control • Process control • Ruler window • System diagnostics control • Parameter set control
		To adapt the column width, tap a column grid line and drag your finger to the right or left.	
		To change the order of the columns, tap a column header and drag your finger to another column header.	
		To move zoomed-in window contents, drag with one finger.	Browser

Icon	Gesture	Behavior	Supported WinCC controls
	Double tap	To edit a cell value, double-tap the cell. Requirement: • Process control: The "Edit" button is pressed. • Parameter set control: A parameter set is selected.	• Process control • Parameter set control
	Scale	To zoom in or out in an operating element, drag with two fingers in the operating element window.	Trend control, browser
	Two-hand operation Hold the release button with one finger, and operate an object with the second finger	An operating element can be configured for two-hand operation, that is, the object can only be operated when a release button is being pressed at the same time. For two-hand operation in WinCC you configure: • A button that is defined as a release button in the security settings of a plant screen. • The security property "Require explicit unlock" at all operating elements that can only be operated when the release button is pressed.	

See also

Special features for multi-touch operation (Page 7222)

Special features for multi-touch operation

Scrolling in lists and controls

You can scroll through lists and controls by dragging.

Special features of the trend view

You enlarge or reduce the view in "Trend view" and "f(x) trend view" objects by pinch-to-zoom with two fingers.

Double tap to switch from the magnified trend view back to the normal view.

The zooming function is limited to the time axis in the "Trend view" object.

If you have enabled the option "Range > Auto-size" during configuration of the value axes in f(x) trend view, the axes are constantly calculated during zooming.

Horizontal scrolling is not supported in the "Trend view" object.

Note

Current view is not persistent

The changes of zoom factor and position changed by scrolling are not saved.

The trend view is reset to the default setting during a screen change.

See also

Supported gestures (Page 7220)

Two-hand operation of operator controls

Two-hand operation of operator controls

Introduction

WinCC supports two-hand operation of operator controls for Unified Comfort Panel. It ensures safe operation of operator controls which are used to change critical system settings, for example, control tags with machine limits.

Locked and unlocked operator controls

You define specific operator controls as "locked operator controls" for two-hand operation of operator controls. Locked operator controls usually cannot be operated in runtime. Operators can only operate the locked operator controls when they press a release button at the same time.

In runtime, locked operator controls can only be accessed with the tab sequence when a release button is pressed at the same time.

Locked operator controls and release buttons

You can configure all operator controls as locked.

You must configure at least one button in the screens as release button. This can be any unlocked button. The unlocking of locked operator controls by pressing the release button has an effect on all open screens.

Display in runtime

The locked operator controls are displayed as dimmed in runtime. The locked operator controls are completely visible when they are unlocked by means of the release button.

Simulation of projects with multi-touch functions

WinCC supports the simulation of configured multi-touch functions. Requirement is that your monitor supports multi-touch operation.

Locking and unlocking operator controls

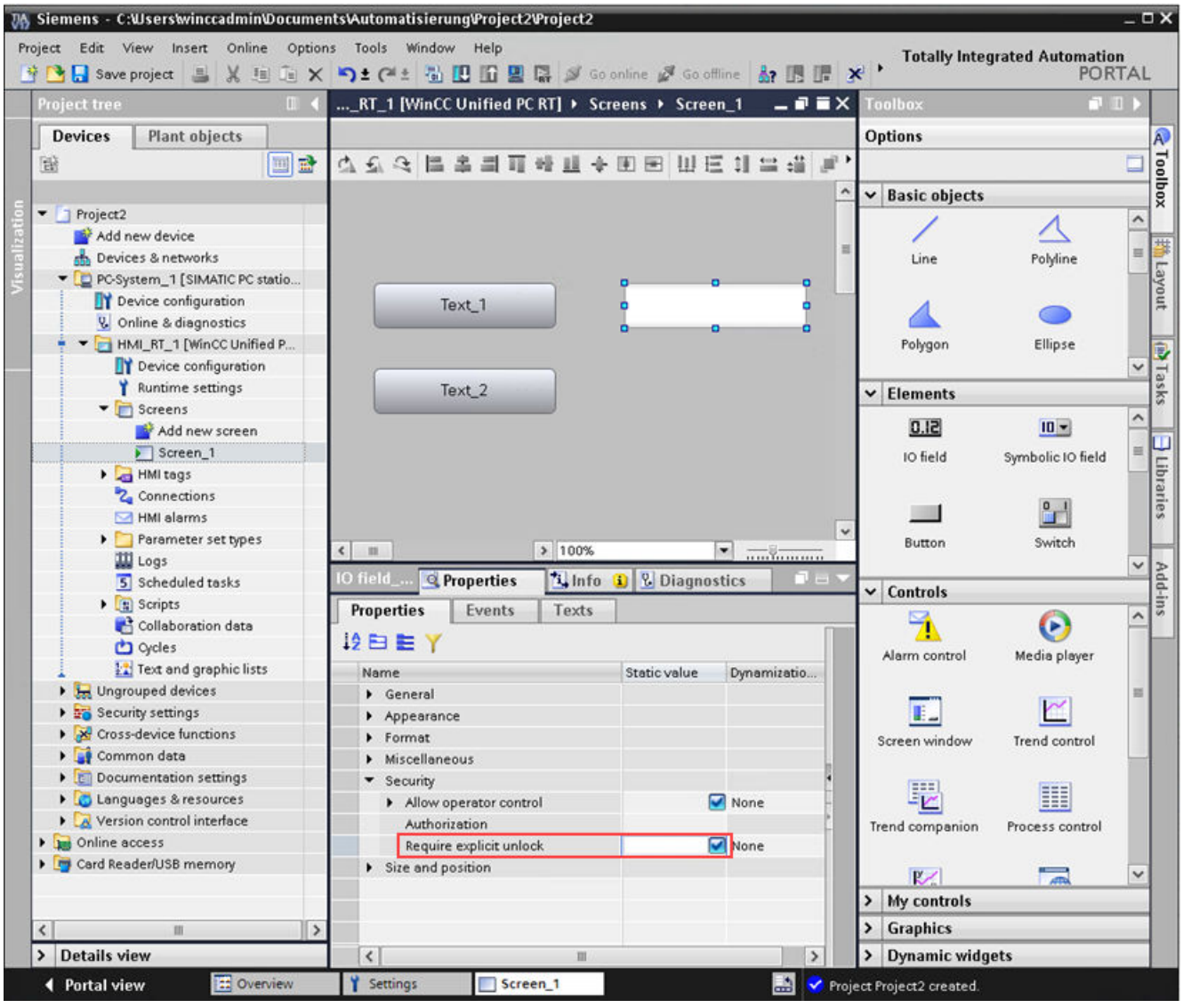
You can lock and unlock operator controls in projects for multi-touch devices. Locked operator controls can only be operated in runtime when the operator presses a release button at the same time.

You can lock and unlock individual operator controls or several operator controls simultaneously.

Procedure

1. Configure operator controls of the type I/O field, button or slider.
2. Select the required operator control(s).

3. To lock the operator controls, enable the "Require explicit unlock" option under "Properties > Properties > Security".



4. To unlock the operator controls, disable the "Require explicit unlock" option under "Properties > Properties > Security".

In runtime, locked operator controls can only be operated when a release button is pressed at the same time.

Note

Locking of operator controls is an add-on to the existing security settings of the operator control. This means that in case of locked operator controls - in addition to pressing the release button - the general operability ("Allow operator control" option) and the required operator control ("Authorization" property) must be present so that the operator control can be operated in runtime.

Defining the release button

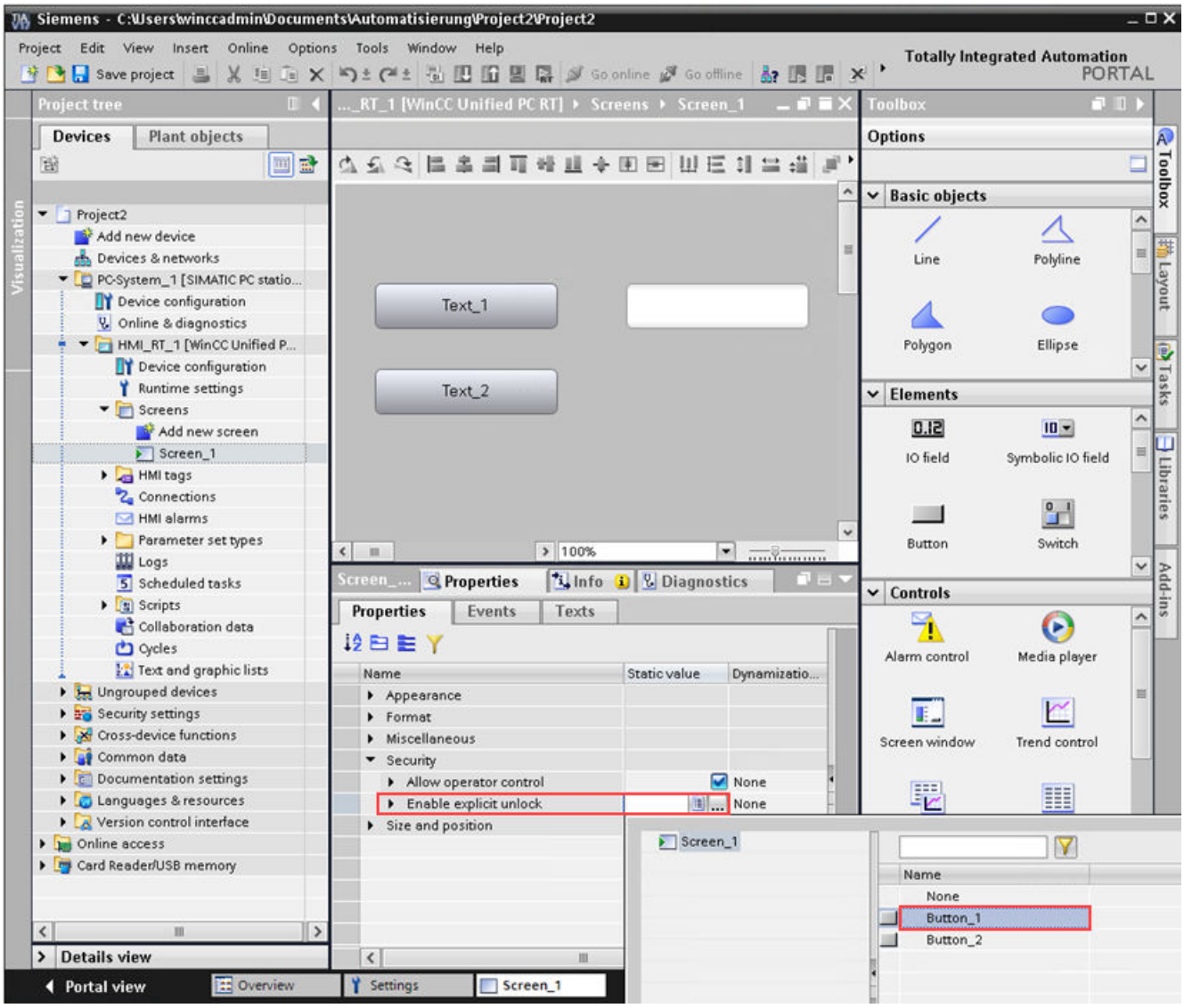
To use the locked operator controls, you must configure at least one release button in one of the displayed screens.

Configuring the release button in the screen

So that you can operate locked operator controls on multi-touch devices, configure a release button.

Procedure

1. Select the screen.
2. Select the desired button of the screen under "Properties > Security" under "Enable explicit unlock".



3. To turn a release button back into a normal button, select a different button or "None" under "Properties > Security" under "Enable explicit unlock".

17.2.3.3 Triggering an action

Introduction

Triggering an action at an operator control can mean the following:

- A command is executed.
  Example: Touch a button to trigger a script or perform a predefined function.
- An object is enabled.
  Example: Touch a table cell to enter a value in a list.

Requirement

- You have navigated to the operator control on which you want to trigger the action.
- The operator control has the focus.

Procedure

- Touch the operator control on the touch screen once or twice in rapid succession.

Result

The following results are possible:

- The requested command is executed.
- The screen keyboard is opened and/or the cursor blinks in the input area of the operator control.
- The element is selected and can be moved.

17.2.3.4 Entering a value

Introduction

Depending on the input format, you enter numeric or alphanumeric values in an input field using the screen keyboard.

Requirement

- The object is an input field or table field.
- The operator control is enabled.

Entering a value

1. Enter the desired value.
2. To confirm the value and exit the field, press the <Enter> key.
3. To discard the value and exit the field, press the <Esc> key.

Result

A value is entered or discarded. You navigate as needed to the next operator control.

For more detailed information, refer to the operating instructions for your HMI device.

17.2.3.5 Moving operator controls

Introduction

You operate movable operator controls of a screen item in Runtime via the touchscreen, such as a slider.

Requirement

- A movable operator control is selected.

Procedure

1. Use a corresponding gesture to move the operator control, e.g. "drag" for a slider.
2. To finish the movement, navigate to another screen object or operator control.

Result

The position of the movable operator control and the display in the screen object have changed.

17.2.3.6 Changing Runtime language

Introduction

The project on the HMI device can be multilingual. A corresponding operating element which lets you change the language setting on the HMI device has been configured in runtime.

The project always starts with the language set in the previous session.

Requirement

- The desired language for the project is available on the HMI device.
- The language switching function is linked to an operating element, for example, to a button.

Selecting a language

You can change project languages at any time. The language-specific objects are displayed in the selected language when the language switching function is called.

You can switch the language in Runtime in one of the following ways:

- Use a configured operating element to switch from one language to the next in a list.
- Use a configured operating element to directly set the required language.

17.2.3.7 Web browser of WebKit engine

Introduction

If the "Browser" object is configured for an HMI device, then the "Browser" operating object is displayed in the corresponding runtime screen. Only the web browser of the WebKit engine is available on HMI devices. This web browser offers many HTML5 features, but no Active X support.

HTML5 functions

The following HTML5 standard functions are fully or partly supported by the Web browser of the WebKit engine:

- Parsing rules
- Elements
- Forms and fields
- Output
- Communication
- User interactions
- Performance
- Security
- History and Navigation
- 2D graphics
- Memory
- Animations
- Web applications
- Files and file systems

Note

Functions not supported in the WebKit engine web browser

- Microdata
- Enter
- Peer to peer
- Position and orientation
- Video, audio
- Responsive images
- 3D graphics
- Streams
- Web components

The following tables show the availability of the HTML5 functions in the web browser of the WebKit engine in detail:

Parsing rules	**Available**
<!DOCTYPE html> triggers the standard mode	Yes
HTML5 tokenizer	Yes
HTML 5 tree building	Yes
Parsing Inline SVG	Yes
Parsing Inline MathML	Yes

Elements	**Available**
Embedded invisible data	Yes
New or modified elements	
Section elements	Yes
Grouping content elements that belong together	Partly
Semantic elements of the text level	Partly
Interactive elements	Partly
Global attributes and methods	
Hidden attributes	Yes
Inserting dynamic markups	Yes

Forms and fields	**Available**
Field types	
type = text	Yes
type = search	Yes
type = tel	Yes
type = URL	Yes
type = email	Yes
type = date	No

Forms and fields	**Available**
type = month	No
type = week	No
type = time	No
type = datetime	No
type = datetime-local	No
type = number	Yes
type = range	Yes
type = color	Yes
type = checkbox	Yes
type = image	Yes
type = file	Yes
textarea	Yes
select	Yes
fieldset	Yes
datalist	Yes
keygen	Yes
output	Yes
progress	Yes
meter	Yes
Fields	
Field validation	Yes
Assignment of forms and controls	Yes
Other attributes	Yes
CSS sectors	Yes
Events	Yes
Forms	
Form validation	Yes

Output	**Available**
Full-screen support	No
Web notifications	Yes

Communication	**Available**
Server-sent events	Yes
Web beacons	No
XML HttpRequest Level 2	
File upload	Yes
Response type	Yes
WebSocket	
Basic Socket Communication	Yes
ArrayBuffer and Blob	Yes

User interactions	**Available**
Drag-and-drop	
Attributes	Yes
Events	Yes
Editing HTML	
Editing elements	Yes
Editing documents	Yes
CSS sectors	No
APIs	Yes
Clipboard	
Clipboard for API and events	No
Spell check	
Spelling attributes	Yes

Performance	**Available**
Native binary data	Yes
Workers	
Web workers	Yes
Shared workers	Yes

Security	**Available**
Web Cryptography API	No
Content Security Policy 1.0	Yes
Content Security Policy 1.1	No
Cross-Origin Resource Sharing	Yes
Cross-Document Messaging	Yes
iFrames	
Sandboxes iFrame	Yes
Seamless iFrame	Yes
iFrame with inline contents	Yes

History and Navigation	**Available**
Session history	Yes

2D graphics	**Available**
Canvas 2D graphics	Yes
2D primitives	
Text input in graphics	Yes
Path input in graphics	No
Drawing an ellipse in graphics	No

2D graphics	**Available**
Drawing a dashed line in graphics	Yes
System focus ring	No
Functions	
Hit testing	No
Aperture mode	No
Formats for image export	
PNG	Yes
JPEG	Yes
JPEG-XR	No
WebP	No

Animation	**Available**
window.requestAnimationFrame	Yes

Web applications	**Available**
Offline resources	
Application cache	Yes
Service workers	No
Content and scheme handlers	No

Memory	**Available**
Key value storage	
Session memory	Yes
Local storage	No
Database storage	
IndexedDB	No
Blob object store	No
ArrayBuffer object store	No
Web SQL database	Yes

Files and file systems	**Available**
Reading files	
Basic support for reading files	Yes
Creating a blob from a file	Yes
Creating a data URL from a blob	Yes
Creating an ArrayBuffer from a blob	Yes
Creating a blob URL from a blob	Yes
Accessing a file system	

Files and file systems	Available
API file system	No
File API: Folders and system	No

Additional functions	Available
Styles	
Style items	No
Scripts	
Asynchronous script execution	Yes
Signaling script errors in Runtime	Yes
Events for script execution	No
Base 64 encoding and decoding	Yes
JSON coding and decoding	Yes
URL API	Yes
MutationObserver	"Yes" (pre-selected)
Promises	No
Page visibility	"Yes" (pre-selected)
Text selection	Yes
Scrolling (Scroll into view)	Yes

17.2.4 Entering barcodes via handheld readers

Introduction

Optical handheld readers enable you to optically identify components, machines and other objects and to transfer the read-out data on your HMI device directly to certain operating objects.

Optical handheld readers capture codes such as two-dimensional data matrix codes, one-dimensional barcodes and postal barcodes.

Supported optical handheld readers can be found at the following entry on the Internet:

FAQ 19188460 (https://support.industry.siemens.com/cs/ww/en/view/19188460)

You can find templates for the settings and instructions on configuration in the manual for your optical handheld reader.

Procedure

You use the connected optical reader to read a code into the object that has the focus.

After the read-in, confirm the value with the Enter key or with the "Suffix - Enter" that you have previously configured in the settings of your optical reader.

Objects for input with optical handheld reader

The following objects support input via an optical handheld reader:

Object	Preconditions for input
IO field Clock	The corresponding data type is selected. The object and the tag length are configured accordingly. The operating object has the cursor focus.
Parameter set control	The parameter set has the cursor focus.
Browser	The operating object has the cursor focus.
Runtime dialogs which support keyboard entry	The dialog is open and the corresponding input field has the cursor focus.
File browser	The field "File path" has the cursor focus.

Result

The code is read and entered into the corresponding input field.

17.3 Operating Unified PC

17.3.1 Basics

17.3.1.1 Process screens

Behavior of process screens

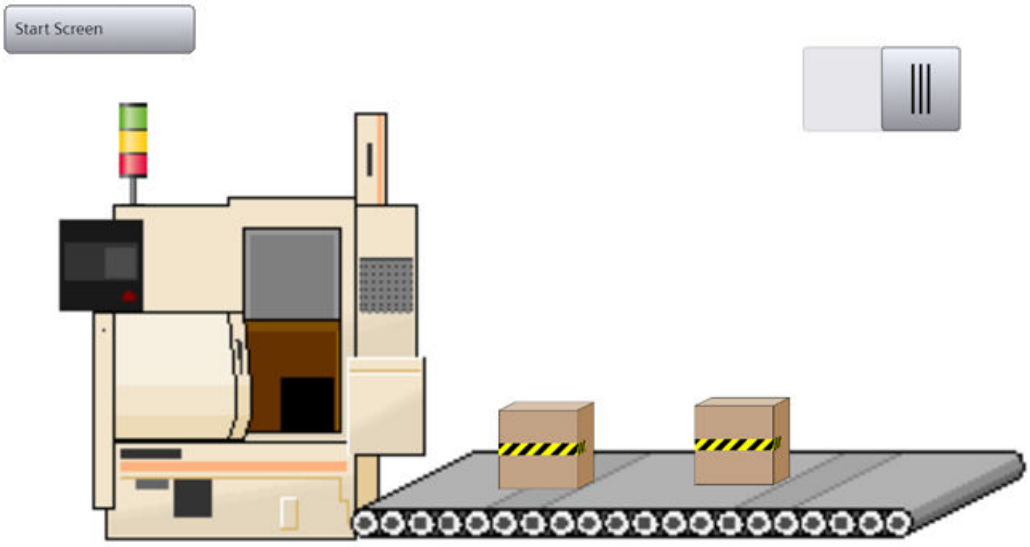
Process screens are static and dynamic representations of the plant, plant units or processes. You use the process screens to operate and monitor the plant or areas within it.

A project on an HMI device consists of multiple process screens. When you start Runtime, the process screen that was defined as the start screen is displayed. You navigate between process screens according to a sequence, navigation or link that was defined by the configuration engineer.

The process screen contains static and dynamic screen objects. Screen objects visualize the current process values from the controller memory and record operator inputs that influence the process. Dynamization is realized through the connection of tags to the screen object during configuration.

Process values and operator inputs are exchanged between the controller and the HMI device by means of tags.

A process screen can be opened and operated by several operating stations simultaneously in Runtime.



Note

Displaying a start screen changed by reloading

A start screen was defined for a project, and the project was started in Runtime. If another start screen is then defined in engineering and the project is loaded into the device again, the last active screen is displayed in Runtime after the connection is established again.

After reloading the project, refresh the screen in Runtime. If your HMI device is a computer, use the F5 key or the browser "Refresh" button to do this.

Screen navigation

Process visualization is generally split between multiple process screens, for example on the basis of functional or technological aspects. Changing between process screens is referred to as screen navigation.

Popup window

With corresponding configuration in the engineering system, clicking on a screen area opens a popup window containing additional information on the screen area.

Example: A screen represents a pump with its valves. When you click on a valve, a popup window opens with detailed information on the valve as well as input fields. You can check the state of the valve in the pop-up window and edit using the input fields.

Predefined styles

The following predefined styles are available for the process screens of the HMI devices:

- Light style
- Dark style
- Expanded style

Note

Compact mode in light and dark style

If the following elements in light or dark style fall below specific dimensions, they are automatically displayed in compact mode:

- Bar
- Slider
- Gauge
- Clock

17.3.1.2 Tags

Behavior of tags

Tags correspond to defined memory areas to which values are written and/or from which values are read. In runtime, tags are output in trends or tables, for example.

External tags correspond to the process values from the memory of an automation system. They are connected to the tags of a connected PLC.

Internal tags transport values within the HMI device. The internal tag values are only available as long as runtime is running.

Value changes to external tags

Value changes to external tags are triggered as follows in runtime:

- By a PLC
  The PLC changes the value of the connected PLC tags.
  During the next update of the external tags, the new value is written to the HMI process image.
- By operator actions or by a script running on the HMI device
  The value change requested in runtime is not directly applied from the HMI process image. Runtime transfers the value to the PLC. The PLC writes the value to the linked PLC tag after successful verification.
  During the next update of the external tags, the new tag value is written to the HMI process picture.

Acquisition mode and acquisition cycle for updating the tags are specified during configuration.

Executing the script of a trigger tag

The script defined for a trigger tag in engineering is executed in Runtime in the following cases:

- During start of Runtime
  The start value of the trigger tag is reported to Runtime.
- When the condition defined for the trigger tag occurs
  For example if the trigger tag changes its state or exceeds a limit value.

Floating point numbers in the web client

Since the web client is implemented via JavaScript, tag values for floating point numbers can only be displayed with a mantissa of up to 54 bits. This leads to rounding of values with a mantissa greater than 54 bit in Runtime.

Note

Values with a mantissa of up to 64 bits are correctly displayed by I/O fields.

Restricted scope of validity "local session"

By default, internal tags apply "system-wide".

As an option, the scope of validity of an internal tag can be limited to "local session". Data related to a session in a multi-user environment is processed independently in each local session.

The use of local session tags is supported in Unified Collaboration and in the web client.

Local session tags permit, for example:

- Individual navigation in screen windows or in different menu structures
- Session-related disabling/enabling of the user
- Session-related position, alignment and rotation of objects in a screen

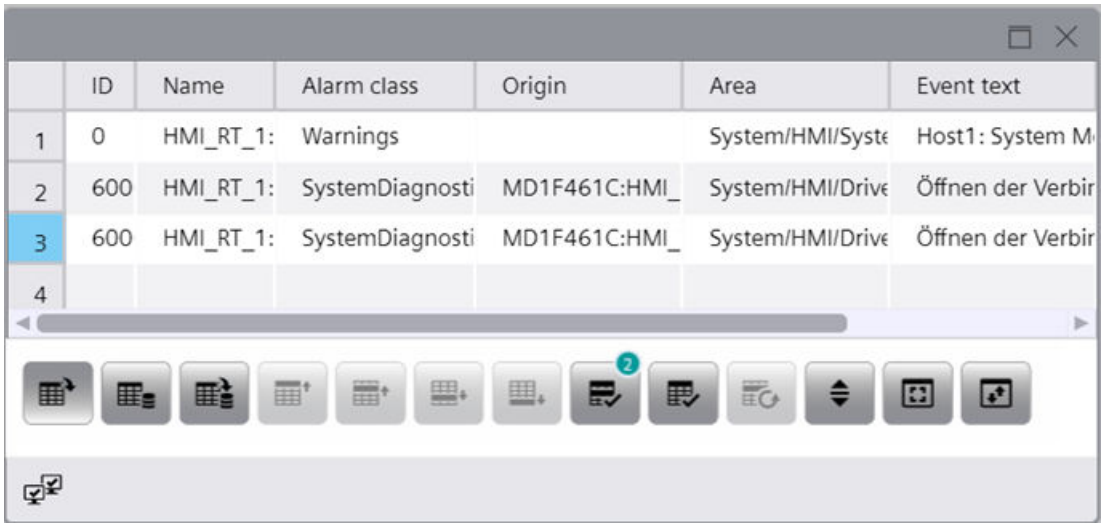
The values of a local session tag are not saved and will be lost at the end of a session.

17.3.1.3 Alarms

Behavior of alarms in Runtime

Depending on the configuration, PLC alarms and HMI alarms from various areas of the plant are displayed in Runtime.

Depending on the configuration, the alarms are labeled according to importance or type and are represented and displayed differently. For example, a pending alarm can be displayed as follows:



17.3.1.4 Logs

Data log

In Runtime, the data logging functions on the server as a log server. On the clients, the data logging functions as a log client. Only the log server accesses the database and compiles and logs the process data. The clients receive log data from the log server.

The log data is visualized in tabular or graphic format on all clients running tag logging in Runtime. The data to be displayed always comes from the log server. All operations on the client are transmitted to the server and the result of the processing is transferred back to the client.

Alarm log

Alarms in the project indicate fault states and operating states of a process. They are generally triggered by the controller. Alarms are displayed on the HMI device in screens. All the data associated with an alarm and configuration data are saved in an alarm log, for example, alarm class, time stamp and alarm text. Each alarm class can be logged separately. Alarms are logged either automatically or by operator intervention.

17.3.1.5 Contexts

Contexts allow you to view plant units according to a certain viewpoint, e.g. according to a certain customer, product, job or shift.

Principle

Contexts always belong to a plant object. They are indicated as follows:

- User-defined contexts:
  Using a program created with the ODK-API
- System-generated contexts:
  For installed Performance Insight and Calendar option packages: Automatically in Runtime
  Example: When a shift starts in Calendar, an archived context value is created with the shift ID

A log entry is generated each time a context (e.g. "Product") is executed. The logged context saves:

- The context value (e.g. "orange lemonade")
- Start time and end time of the execution time
- The quality code

Contexts in the trend control and alarm control

You can filter the content of these controls so that only data that has been generated in a specific plant unit and for the context you have selected is displayed. To do this, select a plant object, a context and one of its logged context values.

Example

A press house produces juices for various beverage brands. Using contexts, employees can display in runtime which alarms have occurred:

- During the production of a specific product (cloudy apple juice, clear apple juice, pear juice etc.).
- For orders for a specific customer (Schmitt, Schulze, Meier).
- During a specific shift (early shift, late shift, night shift).

Contexts in the "Reports" control.

You have the option of linking the generation of reports to the execution of contexts.

If the templates are configured appropriately, the reports available in the control can also contain information about contexts. When a report was generated as an Excel file and reads both contexts and alarms or tag values, you can then use the Excel filter function to filter the alarms and tags by context.

See also

Display context-dependent alarms of a plant object (Page 7291)

Display context data of the plant objects in a trend control (Page 7329)

Adding contexts (Page 7421)

17.3.2 Starting and displaying runtime

17.3.2.1 Internet browsers for WinCC Unified PC

Ensure you have the latest operating system and browser version if you want to access Runtime Unified with this device.

WinCC Unified displays the runtime elements in HTML5. The browser used also has to support this standard. Since the browsers interpret HTML5 differently, it is possible that objects are displayed differently depending on the browser and the browser version used. For example, browsers sometimes display fonts differently.

Compatibility tests were performed for the following browsers. The focus of the compatibility tests was on the browsers marked with *:

Operating system	Browser
Microsoft Windows	• Google Chrome* • Microsoft Edge • Mozilla Firefox, Mozilla Firefox ESR
Android	• Google Chrome* • Firefox • Edge
iOS, Mac	• Safari* • Google Chrome • Firefox • Edge

Browser recommendation

In view of the performance and support of the Runtime standard elements, Google Chrome has proven to be the preferred browser. Its memory requirements are slightly higher than those of the other browsers.

Note

Operating system and browser version

For Runtime operation via Android or iOS, always use the latest operating system.

Use the latest browser version.

Note

Performance differences in different versions of individual browsers

The browser versions can differ from each other, which can result in different behavior regarding the memory requirements and speed.

Note

Suitability for continuous operation

MS Edge and Mozilla Firefox have proven to be problematic in continuous operation.

Known browser problems

The following restrictions apply to the following browsers:

Internet browser	Limitation
MS Edge	• High memory capacity utilization in continuous operation
Mozilla Firefox	• High memory capacity utilization in continuous operation
Mozilla Firefox ESR	• Support of touch gestures for touch panels as of Firefox ESR V59
Google Chrome	• High memory capacity utilization in uninterrupted duty depending on the version. • On Android: Grid lines with a line width ≤1 are not displayed correctly. This is due to the browser's own line thickness representation. As a solution, it is helpful to use a line width ≥1. • No correct representation of elements that use an SVG graphic as background graphic scaled in the Engineering System.

Restrictions to the various functions can also occur, such as the availability of the before and after buttons in the controls.

Current information on browser problems

You can find up-to-date information on display problems in browsers at the Siemens Online Support under the entry ID 109757952.

17.3.2.2 Displaying runtime

Introduction

Use a web browser (web client) to display and operate the Runtime project running on the HMI device. The following options are available to access and display Runtime:

- From the same device (local web client)
  The web browser is installed on the same device as Runtime.
- Remote access from the same network
  The device on which the web browser is installed belongs to the same network as the HMI device.
- Remote access from a foreign network
  The device on which the web browser is installed does not belong to the same network as the HMI device.

Requirement

- The Runtime project is loaded on the HMI device.
- The project runs in runtime.
- The user management configuration of the project is active.
- When using the central user management:
  - At least one user is created in the UMC system.
  - The user created in the UMC system has been imported into the TIA Portal project before the loading.
  - The user has function rights via his/her roles to monitor or monitor and operate the Runtime project.
- When using the local user management:
  - Before the loading, at least one user has been created in TIA Portal.
  - The user has been assigned at least one role. The user has function rights via his/her roles to monitor or monitor and operate the Runtime project.
  - At least one user has the "HMI Administrator" role.

Procedure

1. To view the Unified start page, enter the Unified URL in the browser address bar:
"`https://<IP address of the HMI device or its FQDN or device name>`"
To display the Unified Runtime page directly, append the following string to the URL: "`/WebRH`"
Step 5 is omitted.
Example: "`https://141.73.65.245/WebRH`"

   Note

   Whether you enter the IP address, the FQDN (fully qualified domain name) or the device name in the URL depends on how the web server certificate has been bound to the HMI device. This is defined during Runtime installation or later in the "Website settings" step of WinCC Unified Configuration.

   If needed, ask your administrator.

   The following restrictions apply to entering the URL:

   - FQDN: Only if the HM device belongs to a domain
   - IP address: Not when using dynamic IP addresses
   - Device name: Only when accessing from the same network

   Note

   When using a local web client, you can also enter the "`localhost`" command.

   Example: "`https://localhost/WebRH`"

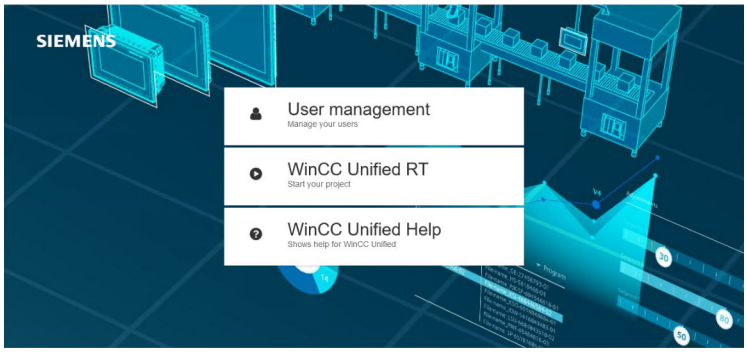
   Regardless of whether the web server certificate is already installed in the browser, you will first see a security warning. Bypass this warning by clicking "Advanced" and "Continue to <link> (unsecure)".

2. Press Enter.
3. If you are accessing the Runtime of the HMI device from this device for the first time and there is no corresponding certificate, install the certificate in the browser. Then reload the page.

4. The start page of Runtime is displayed.

Note

If you experience display problems in the web client, completely delete the browser data (history, form entries, etc.).



If WinCC Unified Online Engineering is installed on the device, the "WinCC Unified Configuration" button also appears.

5. Select "WinCC Unified RT".
The login page of WinCC Unified Runtime is displayed.

Note

After a complete download of a project, an error (SwacLogin) may occur when opening the WinCC Unified Runtime page.

You can find more information at SwacLogin: Errors after complete download (Page 7253).

6. Specify the user name and password of a runtime user.
7. (Optional) Select the language in which runtime is displayed.
8. Confirm your entry.
The Runtime project that is running is displayed in the web browser.

Note

Displayed runtime language

When using central user management, runtime is displayed in the language you have selected in the "User login" dialog during login.

If this language is not available for the current project or if the language setting was not set in the central user management, the following language is displayed:

- Engineering with TIA Portal: The language for which the lowest number was configured in the runtime settings.
- Engineering with Online Engineering: The language set as default language in the "Languages" tab in the "Languages and Resources" editor.

If you do not select a language in the "User login" dialog, runtime is displayed in the language that is set for the browser.

See also

Changing users in runtime (Page 7257)

Starting and stopping a project (Page 7259)

Installing a certificate when accessing via web client (Unified PC) (Page 7247)

Activating user management (Page 7508)

17.3.2.3 Installing a certificate when accessing via web client (Unified PC)

Using root certificates

To enable web browsers to establish a secure connection to WinCC Unified, the root certificate with which the web server certificate of WinCC Runtime was issued must be known in the web browser as a trusted certification authority.

By installing the web server certificate on the PC device, the public root certificate is made available as a download for installation in web browsers on the WinCC Unified home page.

The procedure for installing the root certificate differs depending on your web browser.

Use of self-signed certificates

As an alternative to the root certificate, you can use a self-signed certificate.

NOTICE
Security risk from self-signed certificate A self-signed certificate is not issued by a trusted certification authority. If you use a self-signed certificate from an untrustworthy source, the data transfer is not protected from attacks. Before using self-signed certificates, check the source. Depending on the firewall and network settings, the use of self-signed certificates may be prohibited.

The installation of self-signed certificates is not supported by all web browsers. Depending on the web browser, it is possible to define exceptions.

For more detailed information, refer to the operating instructions of the web browser.

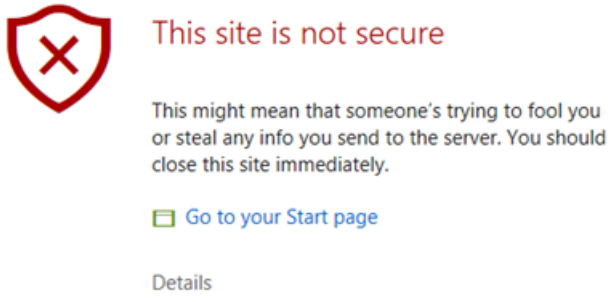
Installing the root certificate for Chrome and Microsoft Edge

Chrome and Microsoft Edge use the Windows system certificate store.

- On devices **with WinCC Unified installation** that have been configured with the Certificate Manager, these web browsers can immediately establish a secure connection to the WinCC Unified web pages because the root certificate has already been installed in the system certificate store.
- On devices **without WinCC Unified Installation** the root certificate must be installed manually.

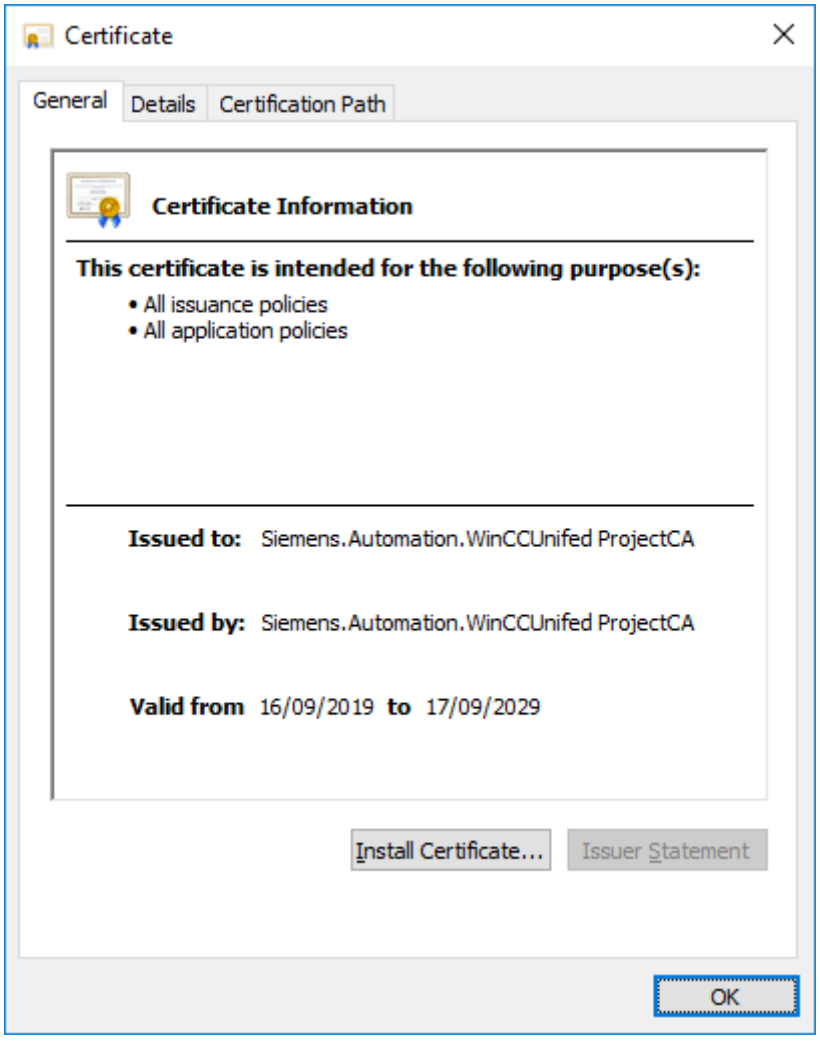
To install manually, follow these steps (for example, Microsoft Edge):

1. Open the WinCC Unified home page via the URL https://<host name>
   At first, an error message appears:



2. Open the field with the error details and confirm that you want to open the web page.
3. On the WinCC Unified home page, select the field "Certificate Authority" and confirm "Open file" in the download dialog.
   The root certificate is downloaded to the default download directory.

4. Open the downloaded file.
   The root certificate is opened with the Windows standard form.



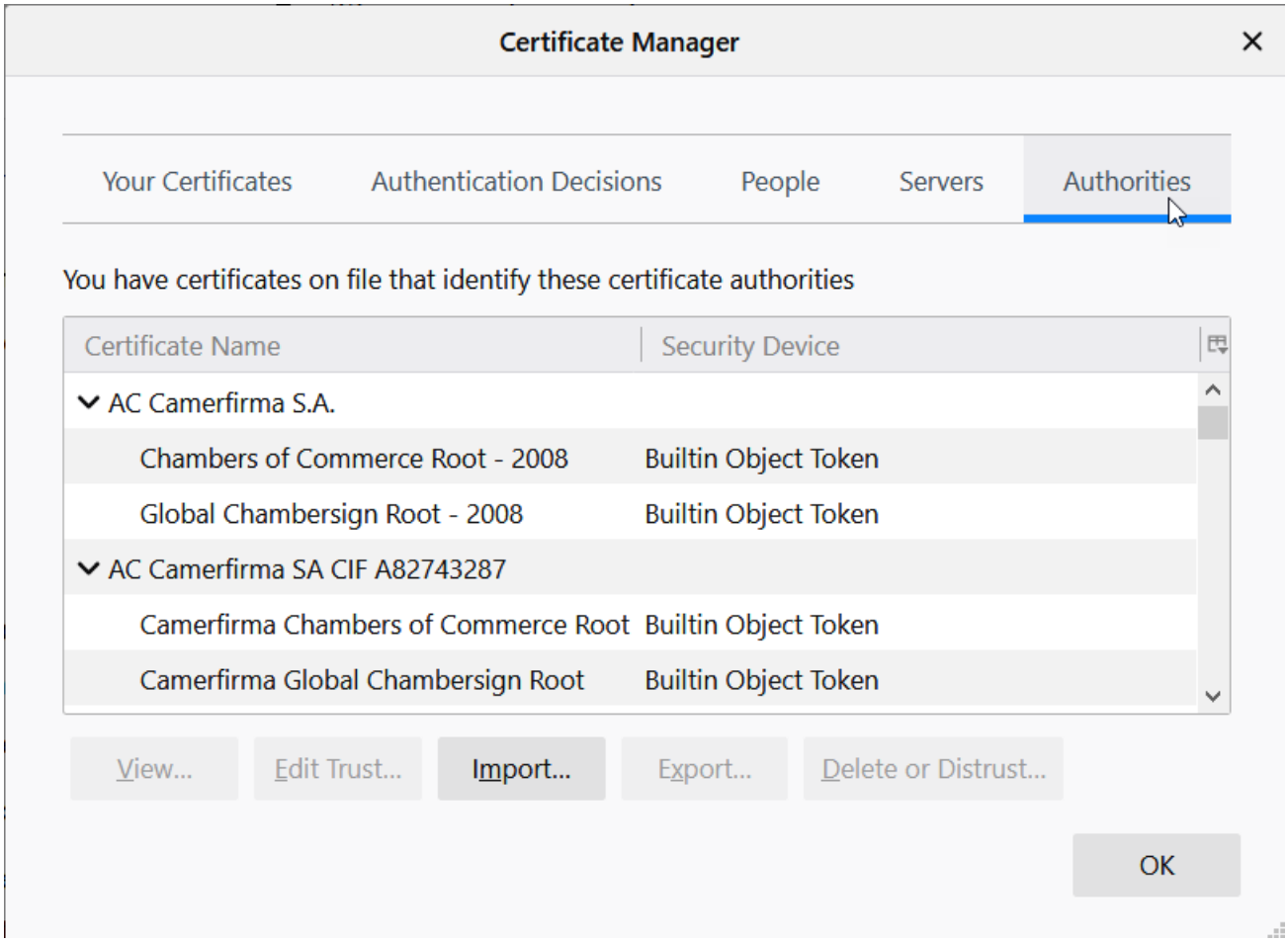
5. To import the root certificate into Windows, select "Install Certificate".
6. In the certificate import wizard, select "Local Machine" as the storage location, "Trusted Root Certification Authority" as the certificate store and start the import process.

Installing the root certificate for Firefox

Firefox uses its own certificate store and must therefore be configured manually on each device once:

1. Open the WinCC Unified home page via the URL https://<host name>
   At first, an error message appears.
2. Open the field "Advanced" and confirm the field "Accept the Risk and Continue".
   An exception is entered for this page in the Firefox certificate management.
3. On the WinCC Unified home page, select the field "Certificate Authority".

4. Save the root certificate. To do this, click "Save file" in the Firefox dialog that follows.
5. Store the certificate in the Firefox certificate store. Proceed as follows:
   - Open the "Settings" page of Firefox.
   - Select "Privacy & Security". There you will find the "Certificates" area further down. Open "Show certificates...".
   - In the "Certificate Management" window, select the "Certification authorities" tab:

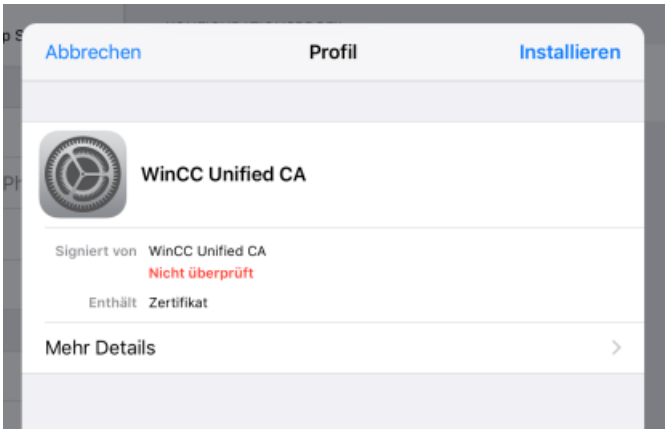


   - Click "Import" and select the root certificate you saved in step 3.
   - In the window that opens, select the option "This certificate can identify websites" and confirm your selection.
     The connection to Runtime is now secure. In the Firefox address bar it is still displayed as unsecure.
   - To show the connection as secure in the address bar, click "Server" and remove the exception created by step 2.

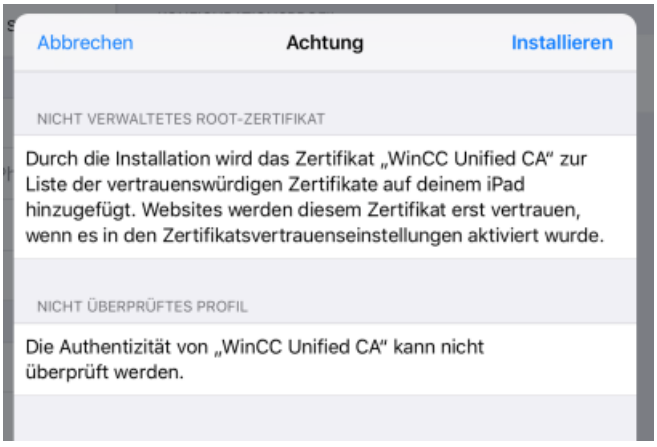
Installing the root certificate on iOS devices

iOS uses its own certificate store and must therefore be configured manually on each device once. An error message also appears when the WinCC Unified home page is opened.

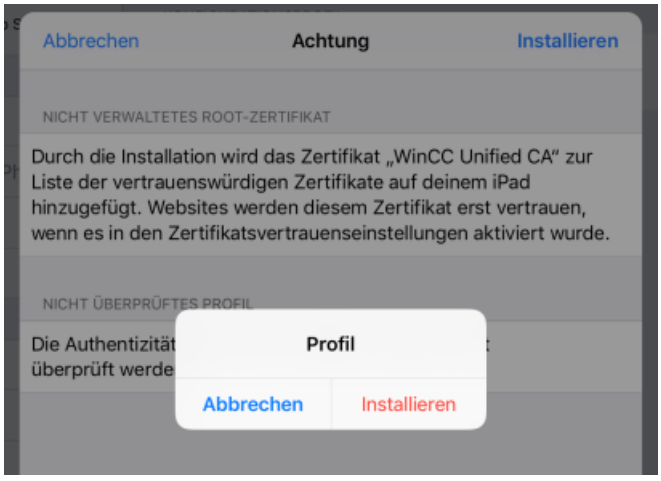
1. Open the field "Advanced" and confirm the field "Accept the Risk and Continue".
2. On the WinCC Unified home page, select the field "Certificate Authority".



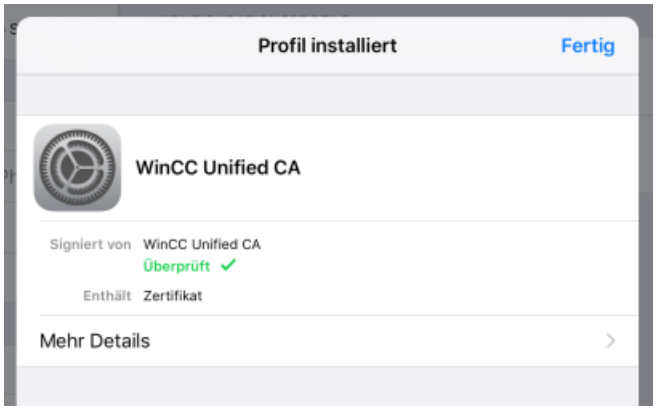
3. Select "Install".



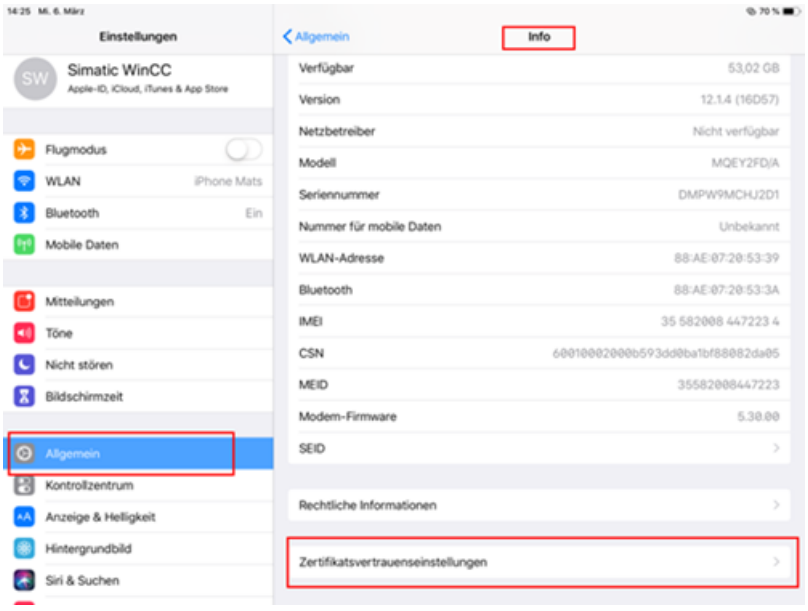
4. Select "Install" again.



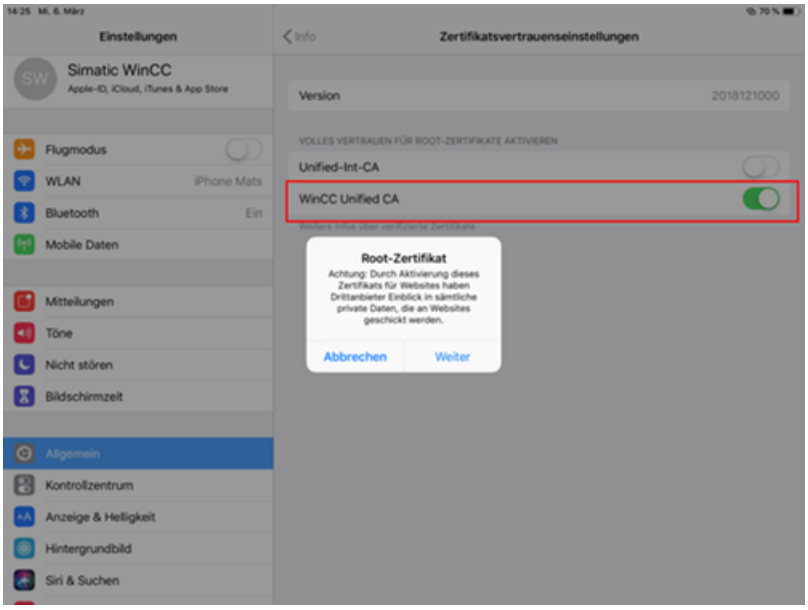
You see the entry "Trusted".



5. Select "General > Info > Certificate Trust Settings".



6. Enable "WinCC Unified CA" and select "Next".



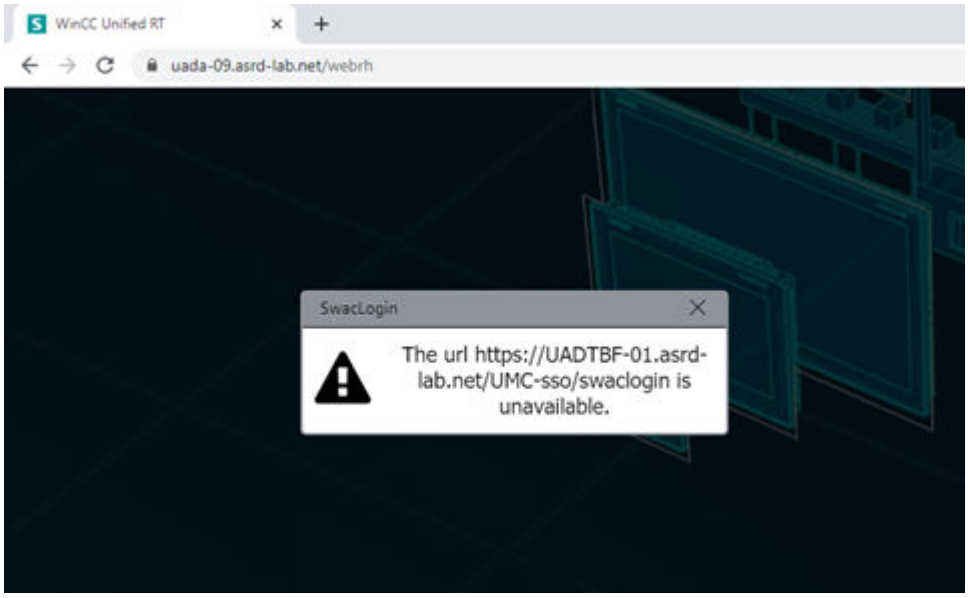
17.3.2.4 SwacLogin: Errors after complete download

After complete download of a project to a Unified PC, an error can occur when you open the WinCC Unified home page. The error can occur regardless of whether you open the home page locally on the PC or from a different device.

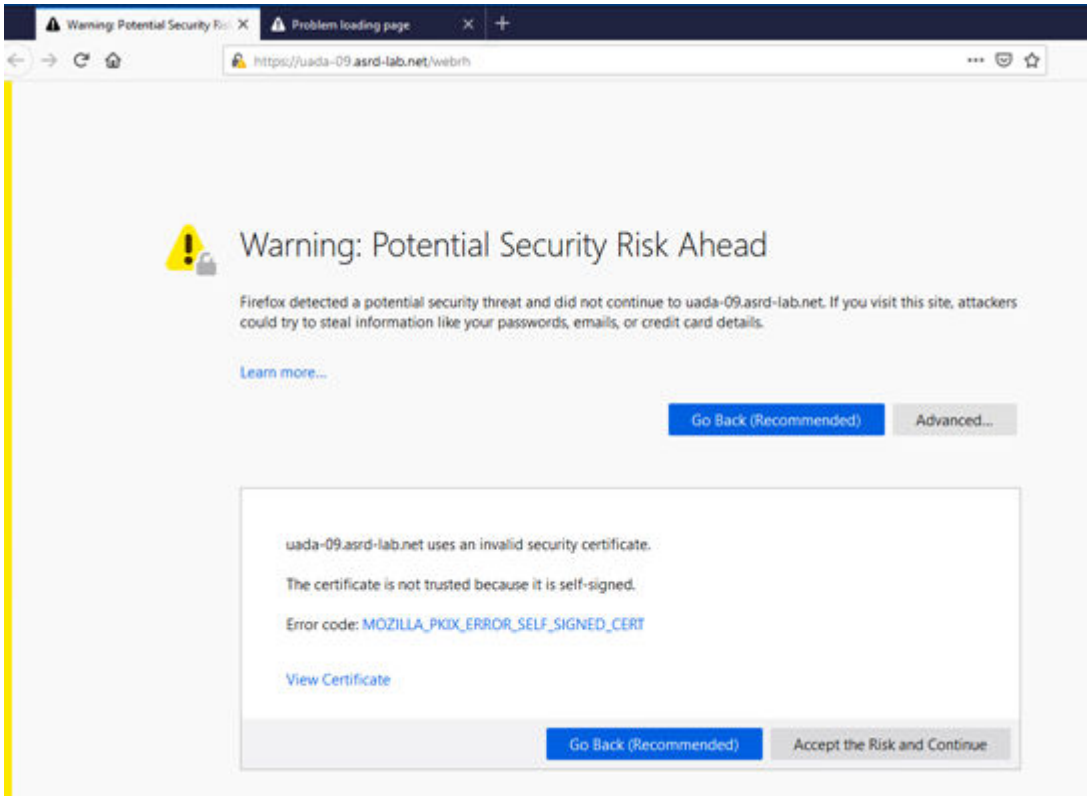
A possible cause of the error is the deletion of the browser cache.

Error description

In "Chrome" and "MS Edge", the error is displayed with the following alarm:



In "Firefox", the error is displayed with the following alarm:

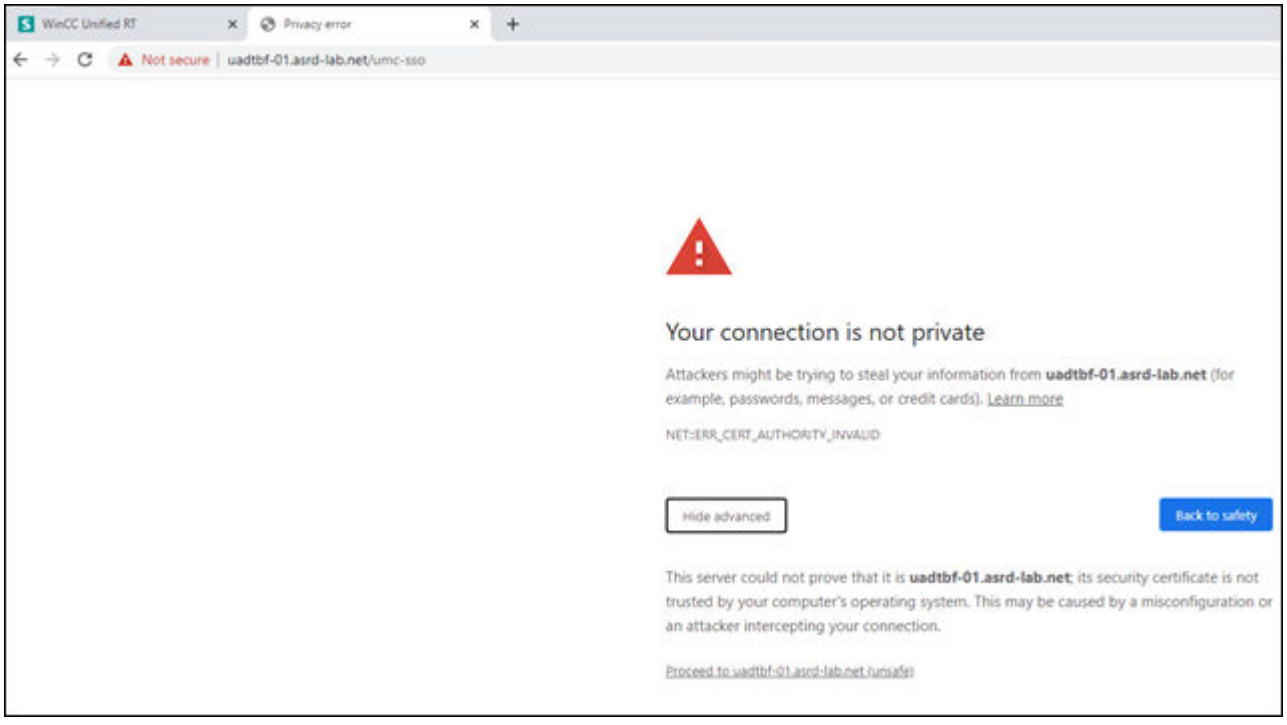


After accepting the warning of a potential security risk, the page remains empty in Firefox. Only the background screen is visible.

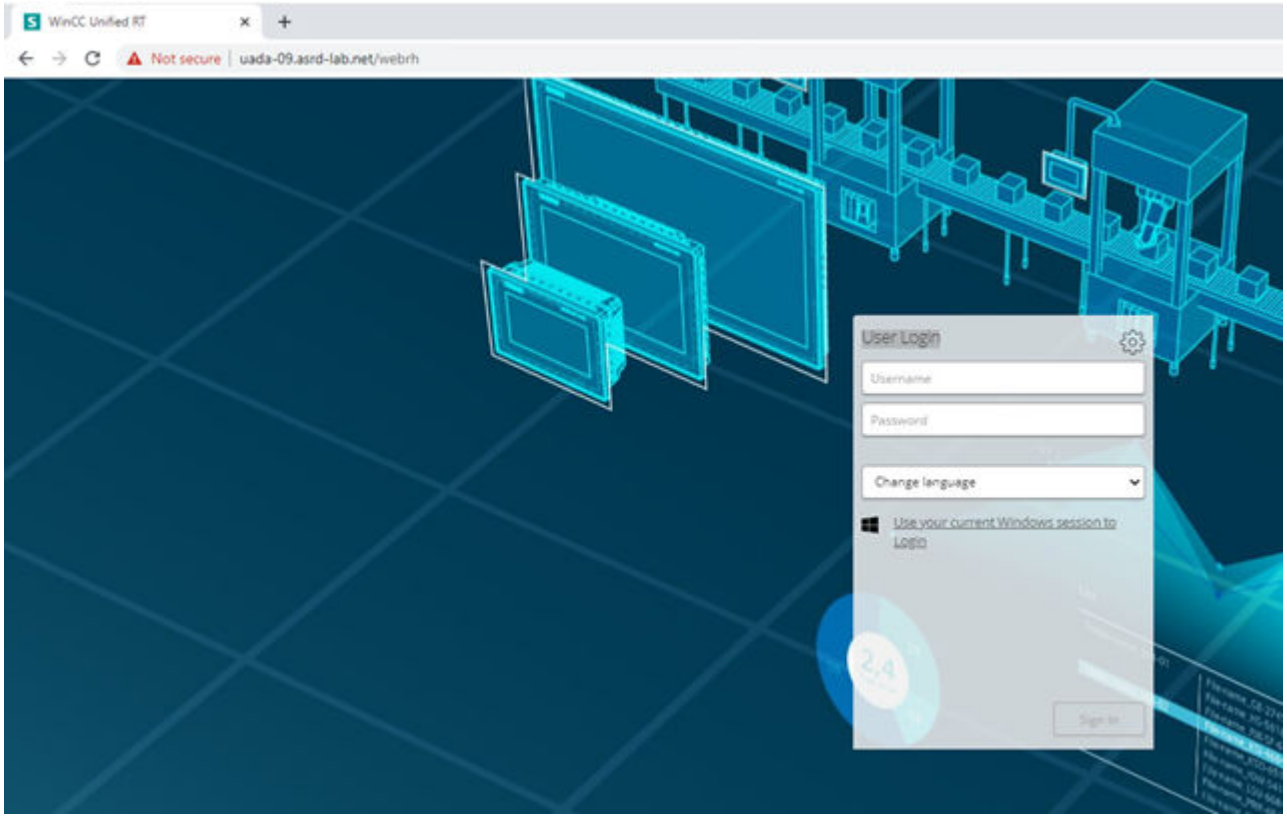
Remedy the error in "Chrome" and "MS Edge"

To fix the error in "Chrome" and "MS Edge", proceed as follows:

1. Open a new tab.
2. Enter the URL address of the identity provider of the UMC server in the address line of the browser. The URL is the same as the one in the error message without "/swaclogin", for example, "https://uadtbf-01.asrd-lab.net/umc-sso".
3. The page with a warning regarding the secure connection is displayed.



4. Accept the warning by clicking on "Proceed to uadtbf-01.asrd-lab.net (unsafe)".
5. The home page with the "User login" dialog is displayed.



Remedy the error in "Firefox"

To remedy the error in "Firefox", follow these steps:

1. Open a new tab.
2. Enter the URL address of the identity provider of the UMC server (ring server) in the address line of the browser, for example, "https://uadtbf-01.asrd-lab.net/umc-sso".
3. A blank page opens. Close the page.
4. Refresh the home page with the function key <F5>. The home page with the "User login" dialog is displayed.

See also

Displaying runtime (Page 7243)

17.3.2.5 Logging out user

If you want to end your Runtime session, you have the following options to log out completely:

- Use the "Logout" system function.
- Log out in the user management.
- Close all instances, i.e. open windows, of the browser in use.

Requirement

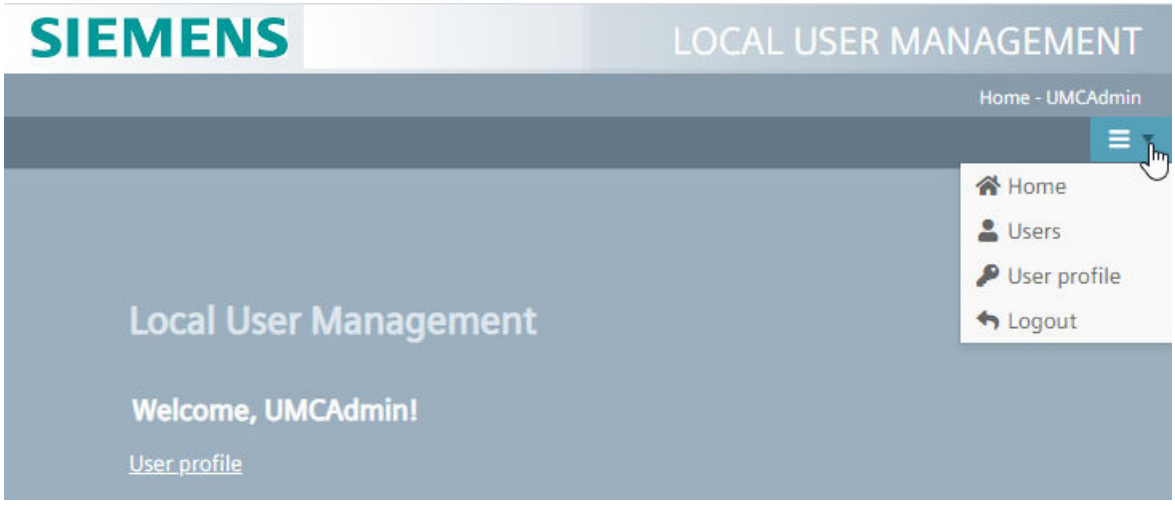
- You are logged in to Runtime.
- When you want to log out in the Runtime project, the system function "Logout" is configured, for example, to the event "Click left mouse button".

Logging out in the Runtime project with the system function "Logout"

- Select the button at which the system function "Logout" is configured.

Logging out in the user management

- Select "Logout" from the menu.
  Your session is ended.



New data downloaded from the TIA Portal is applied during the next login.

17.3.2.6 Changing users in runtime

Introduction

In Runtime, the users that are created in the engineering system can log on.

Requirement

- The IP address or the fully qualified name (name and domain) of the PC on which Runtime is installed is entered in the browser.
  If Runtime is installed on the same PC as the browser, the "localhost" designation can also be used.
- A user is logged into runtime.
  You log in by selecting "WinCC Runtime RT" or "User management".

Procedure

To log off a user and then log on a different user, proceed as follows:

1. Select "User management" on the start page of Runtime.
2. Expand the menu at the top right.
3. Select "Logout".
4. Log in with a different user.

Change logged-on user via RFID card

For local Web clients, WinCC Unified Runtime supports login with RFID and PM-LOGON.

Requirement

- Runtime uses local user management.
- Logon via RFID is active for the HMI device.
  This setting is made during installation of Runtime or later in WinCC Unified Configuration in the "User management" step.
- PM-LOGON is installed on the HMI device.
- The teach-in of the used RFID card with PM-LOGON is completed.
- An RFID reader supported by PM-LOGON is connected to the HMI device.
- The local web client is opened and connected to Runtime.
- A user is logged into runtime.

Procedure

1. Hold the RFID card in front of the reader or insert the card into the reader.
2. If the entry of a PIN was set in PM-LOGON during the teach-in of the card, enter the PIN.

Result

After successful validation of the credentials stored on the card, the user logged-on in Runtime is changed.

If the card requires PIN entry and an incorrect PIN is entered, the previously logged-on user remains logged on.

Note

There is no system feedback about the user change. If required, process screens can be configured in engineering to display the logged-on user.

Note

Additional information on PM-LOGON

See the PM-LOGON user help for information:

- For licensing and installation of PM-LOGON
- To the card readers supported by PM-LOGON
- To the teach-in of the RFID cards.

You can find the PM-LOGON user help at https://support.industry.siemens.com/cs/document/109810587/pm-logon-manual?dti=0&lc=en-DE (https://support.industry.siemens.com/cs/document/109810587/pm-logon-manual?dti=0&lc=en-DE).

17.3.2.7 Starting and stopping a project

A project must be running on the HMI device for Runtime to be displayed.

If no project is running, follow these steps:

1. Start the SIMATIC Runtime Manager tool on the HMI device.
2. Use the tool to get an overview of which projects are loaded on the HMI device.
3. Start the desired project.

You can find more information about the functions of the SIMATIC Runtime Manager and its operation, in the "SIMATIC Runtime Manager" user help.

See also

Exporting tags (Page 7038)

Functions in the SIMATIC Runtime Manager (Page 7491)

17.3.2.8 Switching the Runtime language

Introduction

The project running on the HMI device can be configured in multiple languages. If a corresponding operating element has been configured, you have the option of switching the language set on the HMI device during ongoing operation.

The project always starts with the language set in the previous session.

Requirement

- The desired language for the project is available on the HMI device.
- An operating element is configured, for example, a button that is linked to the language switching function.

Procedure

You can switch between the languages at any time in runtime. Language-specific objects are immediately displayed on the screen in the new language when you switch languages.

Depending on the configuration, you have the following options:

- Use a configured operating element to switch from one language to the next in a list.
- Use a configured operating element to directly set the required language.

17.3.3 Runtime operation

17.3.3.1 Overview

Operating variants

The following operating options for Runtime are available:

- Operation with the touch screen
  The device of the web client has a touch-sensitive touchscreen. Use your finger or a suitable touch pen to operate the touch screen.
- Mouse and keyboard operation
  The device of the web client has a mouse and keyboard.

Adhere to the instructions for operating the device in the operating instructions.

Individually configured operation

The configuration engineer has various options available for setting up operation.

Examples of actions whose execution is always determined on a project-specific basis:

- Screen change
- Reporting
- Change the Runtime language

There are no specific operating elements to execute certain functions. The configuration engineer specifies the project-specific execution. The screen change can be triggered, for example, via a button.

Information on project-specific operations can be found in the system documentation.

17.3.3.2 Operation with the touch screen

Overview of operation with the touch screen

Special features when operating using the touch screen

Operation with the touch screen is characterized by the following special features:

- Enable
  To enable the operating element, use your finger or a suitable touch pen to operate the touch screen. To generate a double-click, touch the operating element twice in rapid succession.
- Value input
  You enter numbers and letters on the touch screen with a screen keyboard.

Input using the screen keyboard

The screen keyboard is displayed when you select a screen object that requires input. The screen keyboard is hidden again when input is complete.

Further information on the screen keyboard can be found in the operating instructions of the HMI device.

Supported gestures

Definition

Various touch gestures are available for Runtime operation on mobile devices. Some touch gestures have different effects in the process pictures than in the controls.

Note

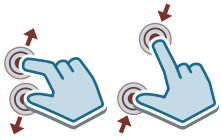
No operation with three or more fingers.

Only use one or two fingers when operating with touch gestures.

If you use more than two fingers with touch gestures, this can cause incorrect operation.

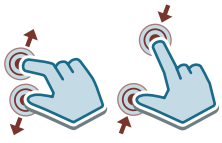
In the case of multi-touch operation with several fingers, you only operate the respectively configured objects.

Supported touch gestures in process pictures

Icon	Gesture	Function
	Tap	To select an object, tap on the corresponding position in the process picture.
	Zoom	To zoom in or out, drag two fingers simultaneously in an area where there are no operating elements. (Zooming) The fingers form the zoom center. With the exception of pop-up windows opened in the process screen, the entire content of the process screen is zoomed.
	Drag with one finger	After zooming in a screen, you can move the screen section. To do this, drag a finger in the desired direction in an area where there are no operating elements. Pop-up windows open in the process screen are not moved with it.
	Keep pressed	To call the shortcut menu, press for longer than a second on the object or the link. The function corresponds to a right-click.

Supported touch gestures in screen windows

The following gestures in the screen window only affect the screen window, not the screen:

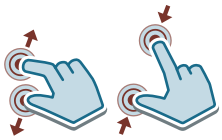
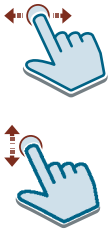
Icon	Gesture	Function
	Drag with one finger	After zooming into a screen window, you can move the screen window section. To do this, drag a finger in the desired direction in an area where there are no operating elements.
	Drag with two fingers	With corresponding configuration in engineering, you can also move the screen window section with two fingers.
	Zoom	With corresponding configuration in the engineering, you can enlarge or reduce the screen window section independent of the zoom factor of the screen. Drag simultaneously with two fingers (zooming). If the screen window section is smaller than the content configured for the screen window due to zooming, the fingers serve as zoom center, otherwise the upper left corner of the screen window.

The following gestures work analogously to the gestures supported in the screen:

- Tap
- Keep pressed

Supported touch gestures in controls

Icon	Gesture	Behavior	Supported WinCC controls
	Tap	• To select a row, tap the row. • With corresponding configuration of the control: To select a cell. • To sort a column, click on the column header. • In trend controls: Zooms into the trend area along the X/Y axis. Requirement: The "Zoom +/-", "Zoom time axis +/-" or "Zoom value axis +/-" button is pressed.	• Alarm control • Process control • Trend control • f(x) trend control • Ruler window • System diagnostics view • Parameter set control
	Tap with two fingers	Zooms out in the trend control. Requirement: The "Zoom +/-", "Zoom time axis +/-" or "Zoom value axis +/-" button is pressed.	• Trend control • f(x) trend control
	Drag with two fingers	To scroll vertically or horizontally in the table of the control, drag in the control window with two fingers in the desired direction.	• Alarm control • Process control • Ruler window • System diagnostics view • Parameter set control
	Drag with one finger	• Moves the ruler. • Moves the x-axis or y-axis. Requirement: The "Move trend area" or "Move axis area" button is pressed or the control is zoomed in.	• Trend control • f(x) trend control
		To select multiple rows, tap a row and drag your finger up or down. With corresponding configuration of the control: To select multiple cells.	• Alarm control • Process control • Ruler window • System diagnostics view • Parameter set control
		To adapt the column width, tap a column grid line and drag your finger to the right or left.	
		To change the order of the columns, tap a column header and drag your finger to another column header.	
	Double tap	To edit a cell value, tap the cell twice. Requirement: • Table view: The "Edit" button is pressed. • Parameter set control: A parameter set is selected.	• Process control • Parameter set control

Icon	Gesture	Behavior	Supported WinCC controls
	Zoom	To zoom in or out in the trend control, drag with two fingers in the control window. Requirement: Trend control is paused and no zoom button is active. Or "Move trend area" is active.	Trend control
	Swiping (horizontally and vertically)	To quickly scroll left or right or up or down within the table of the control, swipe in the corresponding direction.	• Alarm control • Process control • Ruler window • System diagnostics view • Parameter set control

Special features of touch operation

Multi-touch operation of process pictures

WinCC Unified supports multi-touch operation in screens.

Simulation of projects with multi-touch functions

WinCC Unified supports the simulation of configured multi-touch functions. Requirement is that your monitor supports multi-touch operation.

Restrictions for touch gestures

Do not start moving on the following objects with one-finger as well as two-finger gestures:

- Release buttons
- Buttons with configured "Release" or "Press" event.
- "Browser" controls
- Custom web controls
- Touch area
- Elements and controls that manage touch gestures themselves (e.g. sliders and trend control)

Releasing locked operator controls by two-hand operation

Unified supports safe operation of controls that can be used to change critical system settings, such as control variables with machine limits. Such operator controls can be configured as locked.

Locked operator controls are displayed dimmed in runtime. To operate them, simultaneously press the release button provided for this purpose.

Releasing the locked operator controls by pressing the release button has a cross-screen effect on all open screens.

In Runtime, locked operator controls can only be accessed with the tab sequence if a release button is pressed at the same time.

Scrolling in lists and controls

You can scroll through lists and controls by dragging.

Special features of the trend control

You enlarge or reduce the view in "Trend control" and "function trend control" objects by pinch-to-zoom with two fingers.

Double tap to switch from the magnified trend control back to the normal view.

The zooming function is limited to the time axis in the "Trend control" object.

If you have enabled the option "Range > Auto-size" during configuration of the value axes in function trend control, the axes are constantly calculated during zooming.

Horizontal scrolling is not supported in the "Trend control" object.

Note

Current view is not persistent

The changes of zoom factor and position changed by scrolling are not saved.

The trend control is reset to the default setting during a screen change.

17.3.3.3 Triggering an action

Introduction

Triggering an action at an operating element can mean the following:

- A command is executed.
  Example: Click a button to trigger a script or to execute a pre-defined function.
- An object is enabled.
  Example: To enter a value in a list, click in a table cell.

Requirement

- You have navigated to the operating element on which you want to trigger the action.
- The operating element has the focus.

Procedure

- Tap the operating element on the touchscreen once or twice in quick succession.

Result

The following results are possible:

- The requested command is executed.
- The cursor flashes in the input area of the operating element.
  When accessing via touch devices: The screen keyboard opens.
- The element is selected and can be moved.

17.3.3.4 Entering a value

Introduction

Depending on the input format, you enter numerical or alphanumerical values in an input box.

Requirement

- The object is an input field or table field.
- The operating element is enabled.

Entering a value

1. Enter the desired value.
2. To confirm the value, press the <Enter> key or click on a blank area of the screen.
3. To discard the value, press the <Esc> key.

Result

The input is accepted or discarded.

The input box still has the focus.

17.3.3.5 Moving operator controls

Introduction

There are screen objects with movable operator controls, e.g. a slider.

Requirement

- A movable operating element is selected.

Procedure

1. To move the operating element, move it while holding down the mouse button or use a corresponding touch gesture, e.g. "Drag" for a slider.

Result

The position of the movable operating element and the display in the screen object have changed.

17.3.3.6 Placing the focus on objects

You have the following options:

- Click on the object.

  Note

  Giving focus to objects with a transparent background

  If an object has a transparent background, click on a visible area of the object.

- Press <Tab> until the object has the focus.

See also

Operating objects with transparent fill (Page 7267)

17.3.3.7 Operating objects with transparent fill

The objects displayed on a screen can have transparent ranges.
Example: Sliders, bars and pointer instruments are enclosed by a transparent rectangle.

Requirement

An event which is triggered by operating actions such as typing or clicking has been configured for the object in the engineering.

Trigger event

To trigger the event, proceed as follows:

- If the object does not have the focus, click a visible part of the object, e.g. its border.
- If the object already has the focus, the event is also triggered by clicking in the transparent area.

17.3.3.8 Flashing

Flashing in Runtime

You can display the objects flashing in Runtime. Scripts can be used to switch flashing on and off and influence the properties of the flashing.

Configure the flashing behavior of an object property in the engineering system for each color setting of an object that supports flashing.

Note

The flashing in Runtime does not change the color value of the property.

17.3.4 Controls

17.3.4.1 Overview of controls

Runtime has operable controls in process pictures.

The following controls are available depending on the configured access rights:

Icon	Control	Brief description
	Screen window	Displays other screens of the object.
	Trend control	Displays graphical representations of tag values from the current process or from a log in the form of trends with values over time from the controller or a log.
	f(x) trend control	Represents the values of a tag as a function of another tag.
	Browser	Displays HTML pages.
	Media Player	Enables video and audio files to be played.
	Alarm control	Shows currently pending alarms or alarm events from the alarm buffer or alarm log.
	Process control	Represents current or logged process data in a table.
	Trend companion	Displays evaluated data and statistics in a table.
	Parameter set control	Shows the parameter sets with which the PLC is set up for production.
	System diagnostics view	Shows the diagnostic status of multiple PLCs via traffic light SVGs.

Icon	Control	Brief description
	GRAPH overview	Provides an overview of the current status of the configured monitoring.
	PLC code view	Displays the current program status of user programs.

17.3.4.2 Operating alarms

Basics of alarms

Alarm system

Introduction

Alarms show events, operating modes or faults that occur in runtime in the plant.

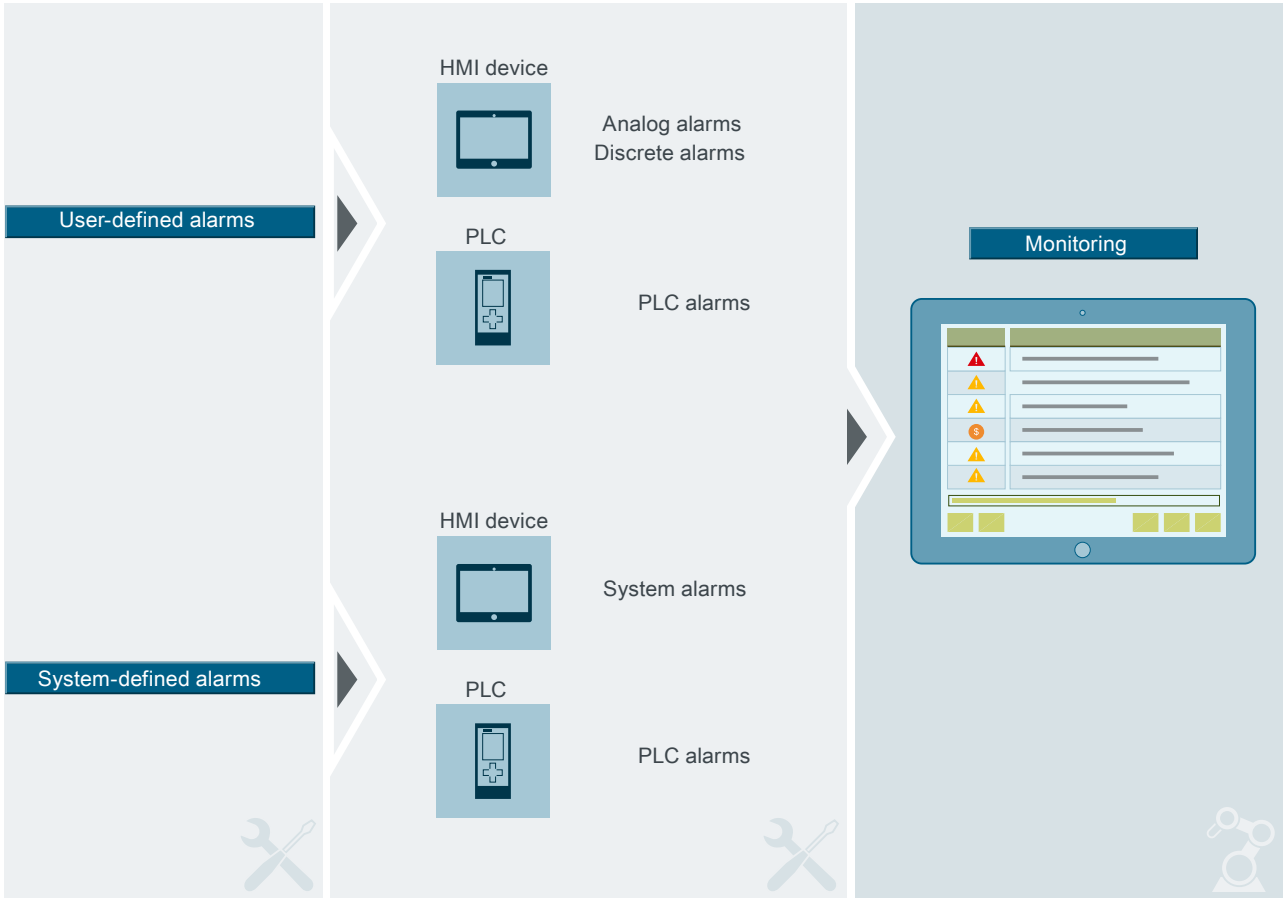
You can use alarms for diagnostic purposes, for example, when troubleshooting. They will help you to immediately locate the cause of the fault. You can adjust your processes through targeted intervention so that compliant products continue to be produced despite the fault, or the process is stabilized, and the fault only causes a minimal loss of production.

The acquired alarms are displayed on the HMI device in screens. The alarm system logs the alarms from the ongoing process. Targeted access to the alarms combined with supplementary information about individual alarms ensures that faults are localized and cleared quickly. This reduces stoppages or even prevents them altogether.

The alarm system in WinCC Unified Scada

The alarm system distinguishes between the following alarms:

User-defined alarms	Analog alarms	Display limit value violations (value changes) They are used to monitor the plant.
	Discrete alarms	Display status changes They are used to monitor the plant.
	User-defined PLC alarms	Displays the status values of the PLC. They are used to monitor the plant and are configured in STEP 7.
System-defined alarms	System alarms	Are included in the HMI device. They are used to monitor the HMI device.
	System-defined PLC alarms	They consist of system diagnostic alarms and system errors. They are used to monitor the PLC. The types of system-defined PLC alarms depend on the PLC used.



See also

Analog alarms (Page 7271)

Discrete alarms (Page 7271)

User-defined PLC alarms (Page 7271)

System alarms (Page 7272)

System-defined PLC alarms (Page 7272)

Alarm control overview (Page 7280)

Alarms

User-defined alarms

Analog alarms

Description

Analog alarms display limit violations. An analog alarm is triggered when the value of the trigger tag meets the trigger condition defined on the analog alarm.

Depending on the selected trigger condition, the alarm is triggered, for example, when the condition value is higher than, lower than, or the same as the defined value.

Example

When the motor speed reaches a critical range as defined in the engineering, an alarm with matching alarm text is displayed. The alarm text can provide the operator with specific instructions on how to check and remedy the situation.

Discrete alarms

Description

A discrete alarm is triggered when the value of a specific bit of a tag changes. The discrete alarms indicate status changes in a plant and are triggered by a controller.

Example

Imagine that the state of a valve is to be monitored during operation. The state of the valve is "open" or "closed".

A discrete alarm is configured for each state of the valve. If the status of the valve changes, a discrete alarm is output, containing for example the following alarm text: "Valve closed".

User-defined PLC alarms

Example of an alarm

"The temperature in Tank 2 is too high."

Description

A user-defined PLC alarm maps the status values of a PLC, for example, time stamp and process values. It is created by a PLC project engineer in STEP 7.

The PLC alarms configured in STEP 7, are applied into the integrated WinCC operation as soon as a connection is established to the PLC.

Note

Automatic update of new or modified PLC alarms on the HMI device

If PLC alarms are configured in STEP 7 and an HMI connection to a SIMATIC S7-1500 controller (firmware version 2.0 or higher) is established, and the PLC and HMI device are configured accordingly in the engineering, the PLC alarms are sent to the HMI device and updated automatically. You can find more information in the TIA Portal help for WinCC Unified.

Note

WinCC only supports PLC alarms of a SIMATIC S7-1500 controller. In addition, WinCC only supports PLC alarms that are automatically updated by the central alarm management in the PLC.

System-defined alarms

System alarms

Description

A system alarm indicates the status of the system and communication errors between the HMI device and the system. System alarms are output in runtime in the configured alarm control. System alarms are output in the language currently set on your HMI device.

The time format (AM/PM or 24-hour format) is based on the selected language. If no translation of the alarm texts exists in this language, English is used as replacement and the corresponding time format is displayed.

Example of an alarm

"Memory is full!"

System-defined PLC alarms

Note

Device dependency

System-defined PLC alarms are not available for all HMI devices.

Description

System-defined PLC alarms are installed with STEP 7 and are only available if WinCC is operated in the STEP 7 environment.

System-defined PLC alarms are used to monitor states and events of a PLC. System-defined PLC alarms consist of system diagnostic alarms and system errors (RSE)

Note

Automatic update of system diagnostic alarms on the HMI device

When an HMI connection to a SIMATIC S7-1500 controller (firmware version 2.0 or higher) is established, and the PLC and the HMI device were configured accordingly in the engineering, the system diagnostic alarms are sent to the HMI device and updated automatically. You can find more information in the TIA Portal help for WinCC Unified.

Note

Note the following restrictions:

- WinCC only supports system diagnostic alarms of a SIMATIC S7-1500 controller.
- WinCC only supports system diagnostic alarms that are automatically updated by the central alarm management in the PLC.

Example of an alarm

"CPU maintenance required"

Alarm blocks

Overview

In Engineering you configure which columns you can see in runtime and which alarm blocks the columns are evaluating. The following section provides an overview of some important alarm blocks.

Example of alarm blocks output in runtime:

Alarm class	Alarm number	Time of occurrence	State machine	Alarm text	Information	Value	Limit value
Warning	1	08/27/2017 11:09:14 AM	Alarm with single-mode acknowledgment	Maximum speed reached	This alarm is...	50	27

Alarm class

The alarm class controls, for example, the display and the acknowledgment concept of the alarm.

Alarm number

You identify an alarm by its number (ID).

Change the alarm number, if necessary; for example, with a consecutive alarm number to mark alarms that belong together in your project.

Note

The alarm number of a system alarm has a higher priority than the number of a user-defined alarm. Do not use numbers that are used by system alarms for user-defined alarms.

Alarm blocks with time and date

These alarm blocks show the time at which the alarm was active, acknowledged or became inactive, etc.

Note

Time zones

By default, the time stamp in the alarm display is converted into the time zone used by the HMI device.

If the alarm display is configured accordingly, the time stamp can be converted to another time zone in Runtime via the "Time base configuration" button.

State machine

An alarm has the state machine or the acknowledgment concept of the alarm class.

The state machine is the way an alarm is displayed in various states and processed by the system.

See section Acknowledgment model (Page 7276).

Alarm state

An alarm always has a specific alarm state in runtime, for example, active or active/ acknowledged. Based on the alarm state, you can understand the process that the alarm went through.

Alarm text

The alarm text (event text) describes the cause of the alarm.

The alarm text can contain output fields for current values. The value is retained at the time at which the alarm status changes.

Priority

Displays the priority of individual alarms.

Note

A priority configured on the alarm has precedence in runtime over the priority configured on the alarm class.

Limit value

Analog alarms display limit violations. Depending on the configuration, WinCC outputs the analog alarm as soon as the trigger tag exceeds or undershoots the configured limit value.

Computer

Operator input alarms have the "Computer" column in the alarm summary. The computer name is displayed for local alarms and the IP address for alarms from the web client.

Users

When an empty user name is passed to an alarm, it will display its formatting string instead of the user name, for example, "@S2%s@".

Duration

Returns the time interval in nanoseconds between triggering of the alarm and its previous status change.

See also

Alarm classes (Page 7275)

Alarm classes

Introduction

Many alarms occur in a plant; these are all of different importance. To make it clear to you, which alarms are most important, alarms are assigned to alarm classes.

Assignment of the alarms to alarm classes and configuration of the alarm classes takes place in the engineering system.

Purpose

The alarm class of an alarm defines the following:

- State machine/acknowledgment model
- Appearance in alarm control (e.g. color)
- Priority

  Note

  The priority configured on the alarm has precedence in runtime over the priority configured on the alarm class.

- Logging

Examples of how to use alarm classes

- The alarm class of the alarm "Fan 1 speed in upper tolerance range" is "Warning". The alarm is displayed with a yellow background. The alarm does not require acknowledgment.
- The alarm "Speed of fan 2 has exceeded upper warning range" is assigned to the "Alarm" alarm class. The alarm is displayed with a red background and flashes at high frequency in runtime. The alarm is displayed until you have acknowledged it.

User-defined and predefined alarm classes

The alarms use user-defined or predefined alarm classes:

- Number and configuration of user-defined alarm classes depend on the configuration of the runtime project in the engineering.
- The number and configuration of predefined alarm classes are provided by the system. You can find more information on predefine alarm classes in the TIA Portal help for WinCC Unified as well as in the user help for WinCC Unified Online Engineering.

See also

Acknowledgment model (Page 7276)

Logging basics (Page 7298)

Acknowledgment model

Introduction

The state machine of an alarm class regulates which statuses the alarms of this alarm class can have. From this it is derived which events can occur for them and whether and how they are acknowledged.

State machines

The following state machines are available for HMI alarm classes:

State machine	Description	Use in predefined alarm classes
Active	Alarm without inactive state without acknowledgment This alarm is displayed until the event that triggered the alarm is no longer pending. The alarm will then no longer be displayed in the alarm control.	Information OperatorInputInformation SystemInformation
Active and inactive	Alarm without acknowledgment This alarm becomes active and inactive without having to be acknowledged.	Notification SystemNotification
Active, requires acknowledgment	Alarm without inactive state with acknowledgment This alarm must be acknowledged as soon as the event that triggers the alarm occurs. The alarm is pending until it is acknowledged.	AlarmWithoutClearEvent SystemAlarmWithoutClearEvent SystemWarningWithoutClearEvent
Active and inactive, requires acknowledgment	Alarm with a single acknowledgment This alarm must be acknowledged as soon as the event that triggers the alarm occurs. The alarm is pending until it is acknowledged and inactive.	Alarm Critical OperatorInputRequest SystemAlarm SystemWarning Warning
Active and inactive, requires acknowledgment and reset	Alarm with acknowledgment and confirmation The alarm requires acknowledgment once the event that triggers, the alarm has occurred, or the alarm has been reset. In addition, the alarm requires confirmation when the event that triggers the alarm is no longer pending. The alarm is pending until it is acknowledged and confirmed.	AlarmWithReset CriticalWithReset WarningWithReset

The following state machines are available for HMI alarm classes that are associated with common alarm classes:

- Alarm with a single acknowledgment
- Alarm without acknowledgment

Alarms requiring acknowledgment

Alarms that indicate critical or hazardous states in the process must be acknowledgeable. Every change in the status of an alarm that requires acknowledgment is logged.

By acknowledging an alarm, you confirm the processing of the state that triggered the alarm. You acknowledge the alarm using the buttons in the alarm control.

Acknowledgment and confirmation of alarms

- Group acknowledgment of alarms in the alarm control
  With the "Group acknowledgement" button you acknowledge all alarms pending in the alarm control that are visible and require acknowledgment.
- Single acknowledgment of alarms in the alarm control
  With the "Single acknowledgment" button you acknowledge a single alarm that is selected in the alarm control.
- Single acknowledgment of alarms with acknowledgment and confirmation in the alarm control
  With the "Single acknowledgment" button you acknowledge a single alarm with the state machine "Alarm with acknowledgment and confirmation" after it was previously acknowledged via group acknowledgment or single acknowledgment and was inactive.

Note

If the "Show recent" button is active, the most recent alarm is always displayed first and is selected.

Group acknowledgment is only carried out for the visible alarms.

See also

Alarm states (Page 7278)

Alarm states

Description

Each alarm has an alarm state. Alarm states are made up from the following events:

- **Active**
  The condition for triggering an alarm is fulfilled. The alarm is displayed, such as "Boiler pressure too high".
- **Inactive**
  The condition for triggering an alarm is no longer fulfilled. The alarm is no longer displayed as the boiler was vented.
- **Acknowledged**
  The operator has acknowledged the alarm.

The alarm state of an alarm at any given time depends on the following factors:

- Which state machine has its alarm class.
  The state machine of an alarm class regulates which events can occur for alarms of this alarm class. From this it is derived which states the alarms can have and whether and how they are acknowledged.
  For an overview of the available state machines, see Acknowledgment model (Page 7276).
- Which events occurred for the alarm.

Note

The display texts of the alarm states are different depending on the language and configuration. The texts displayed in runtime can deviate from the texts shown here.

Alarms without acknowledgment

The following table shows the alarm states for alarms without acknowledgment:

Icon	Alarm state	Description
	Active	The condition of an alarm is fulfilled. The alarm does not need to be acknowledged.
	Inactive	The condition of an alarm is no longer fulfilled. The alarm is no longer pending.

Alarms with acknowledgment

The following table shows the alarm states for alarms with acknowledgment:

Icon	Alarm state	Description
	Active	The condition of an alarm is fulfilled.
	Active/inactive	The condition of an alarm is no longer fulfilled. The operator has not acknowledged the alarm.
	Active/inactive/acknowledged	The condition of an alarm is no longer fulfilled. The operator has acknowledged the alarm after this time.
	Active/acknowledged	The condition of an alarm is fulfilled. The operator has acknowledged the alarm.
	Active/acknowledged/inactive	The condition of an alarm is no longer fulfilled. The operator acknowledged the alarm while the condition was still fulfilled.

Disabled alarms

Operators disable an alarm to, for example, prevent a nuisance alarm from impairing the effectiveness of the system.

- Disabled: The alarm has been deactivated (locked). The alarm transitions to its final state without any further state transitions.
- Not disabled: The alarm is activated (enabled). The alarm is visible again in its last state.

Shelved alarms

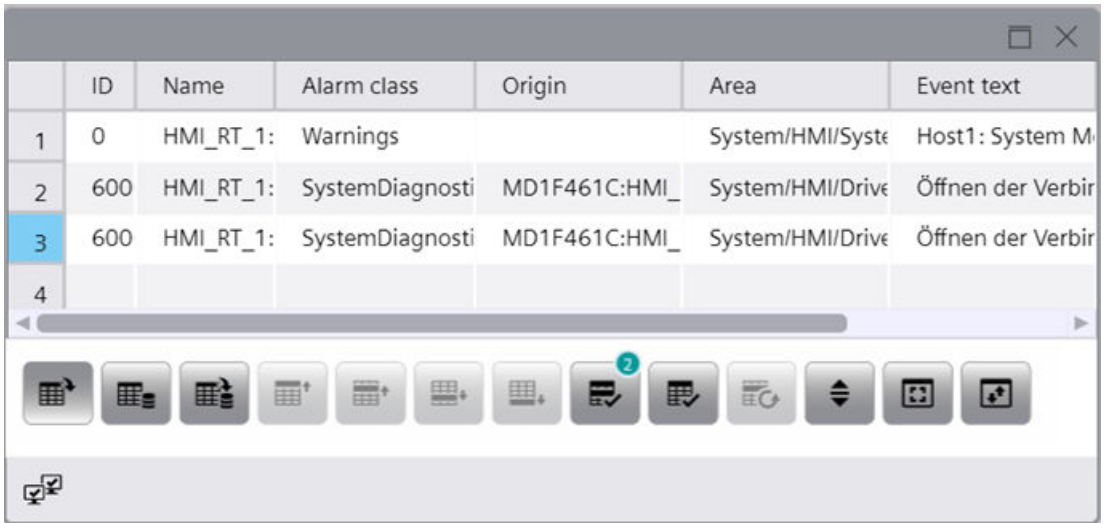
The display of specific alarms is shelved (suppressed) in order, for example, not to overload the operator with information. Manually reset: The alarm was manually reset by the operator. Shelved due to design: The alarm was automatically shelved due to a condition and is automatically hidden in runtime.

Alarm control overview

Introduction

The alarm control displays PLC alarms and HMI alarms that occur during the process in a plant. The alarm control helps prevent faults in the plant or localize and remedy the causes of a fault.

User interface



① Columns for the output alarm blocks
② Alarm summary
Each alarm is displayed in a separate line.
The alarms that are displayed depend on the view or list selected and whether filters are applied.
③ Toolbar for operating the alarm control
④ Information bar

Note

Selection of alarm blocks, column titles and localization depend on the configuration in engineering.

Note

An alarm appears in the alarm control with the date and time stamp crossed out in the following situations:

- A disabled alarm is enabled again.
- An alarm is reloaded after a power failure. This applies only to chronological alarming.
- The automation system is restarted. This applies only to chronological alarming.

Views and lists

Depending on the configuration of the alarms and the situation in the system, a large number of alarms can occur in runtime.

The alarm control offers various views and alarm lists that filter the alarm summary and thus provide a better overview:

View	Description		
Alarm view	Shows the alarms of the currently selected alarm list. Available alarm lists:		
		Display active alarms	Shows the pending alarms. If the toolbar is configured accordingly, you use the "Display options setup" button to set the alarms that are displayed in this list. Default setting: Displays all alarms that are not suppressed.
		Show logged alarms	Shows the logged alarms. The display is not updated immediately when new incoming alarms occur.
		Show and update logged alarms	Updates the logged alarms and shows them. The display is updated immediately when new incoming alarms occur.
		Display defined alarms	Displays the defined alarms. If the toolbar is configured accordingly, you use the "Display options setup" button to set the alarms that are displayed in this list.
Alarm statistics		Displays statistical calculations of logged alarms.	

You enable a view or list using the corresponding button in the toolbar.

Information bar

The information bar shows the different states related to the alarm servers. The information bar contains the following icons:

Icon	Meaning
	Shows the status to the alarm servers: No faulty connections
	Shows the status to the alarm servers: Faulty connections
	Shows the status to the alarm servers: All connections are faulty

With the corresponding configuration in engineering, the information bar shows the number of alarms that are not acknowledged in runtime. The counter includes all connected servers, but no filters.

When a context is selected, the information bar shows the values of the selected context.

Icons for the alarm state

The column for the alarm state can contain the following icons:

Icon	Meaning
In "Show and update logged alarms" list:	
	Alarm is active
	Alarm is inactive
	Alarm acknowledged
In the other lists:	
	Alarm is active
	Alarm is active/inactive
	Alarm is active/acknowledged

Performance data for SIMATIC Unified PC

Number of controller alarms	160000
Number of OPC UA A&C alarms	20000
Number of alarms per second (continuous load)	20
Number of pending alarm events	Unlimited
Number of alarms per 10 seconds (alarm burst)	8000

The maximum number of alarms that can be displayed in Runtime depends on the selected view:

View	Maximum number of alarms that can be displayed.
Display active alarms	No limit
Display defined alarms	
Alarm statistics	
Show logged alarms	1000
Show and update logged alarms	100

See also

Buttons of the alarm control (Page 7283)

Buttons of the alarm control

You operate the alarm control using the buttons on the toolbar. The buttons that are available depend on the configuration:

Button		Description
	Display active alarms	Displays the currently pending alarms. With the ""Active alarms" setup" button, you set which alarms belong to the active alarms.
	Show logged alarms	Shows the logged alarms. The display is not updated immediately when new incoming alarms occur.
	Show and update logged alarms	Updates the logged alarms and shows them. The display is updated immediately when new active alarms occur.
	Alarm statistics	Visualizes statistical information of logged alarms, such as frequency and display duration.
	Alarm statistics setup	Setting options for calculating the alarm statistics.
	Display defined alarms	Shows the alarms configured in the system.
	Alarm signaling equipment	Shows all alarms for which the alarm signaling equipment was configured. The alarm signaling equipment is a visual or acoustic signal, such as a horn or warning light, that is displayed in addition to the alarm control in the system.
	First line	Selects the first of the displayed alarms. The visible area of the alarm control is moved, if required. This button is only operable if the "Show recent" function is disabled.
	Previous line	Selects the previous alarm, starting from the currently selected alarm. The visible area of the alarm control is moved, if required. This button is only operable if the "Show recent" function is disabled.

Button		Description
	Next line	Selects the next alarm, starting from the currently selected alarm. The visible area of the alarm control is moved, if required. This button is only operable if the "Show recent" function is disabled.
	Last line	Selects the last of the displayed alarms. The visible area of the alarm control is moved, if required. This button is only operable if the "Show recent" function is disabled.
	Skip to the next alarm that requires acknowledgment	Selects the next alarm that requires acknowledgment, starting from the currently selected alarm. The visible area of the alarm control is moved, if required. This button is only operable if the "Show recent" function is disabled.
	Previous page	Navigates to the previous page.
	Next page	Navigates to the next page.
	Single acknowledgment	Acknowledges the selected alarm. If using the multiple selection, the selected alarms which require single acknowledgment are not acknowledged. A counter shows how many alarms are not acknowledged. The counter includes all connected servers, but no filters.
	Group acknowledgment	Acknowledges all pending, visible and acknowledgeable alarms in the alarm control if they are not individually acknowledgeable.
	Single confirm	Resets the alarm. Relevant for alarms with the state machine "Alarm with acknowledgment and confirmation", which were already acknowledged and inactive.
	Show recent	Defines whether it is always the latest alarm that is selected in the alarm control. Button not pressed: The "Show recent" function is active. • The most recent alarms are always shown first in the alarm control. Alarms that have been filtered out of the alarm control are not displayed. • The visible area of the alarm control moves automatically, if necessary. • You cannot select the alarms individually or sort them by column. Button pressed: The "Show recent" function is paused.
	Info text setup	Opens a dialog that shows a help text configured for the selected alarm.
	Comments setup	Opens a dialog for adding a comment.
	Disable alarm	Disables an alarm.
	Enable alarm	Enables a disabled alarm.
	Shelve alarm	Resets an alarm, for example, to prevent a nuisance alarm from impairing the effectiveness of your system.

Button		Description
	Unshelve alarm	Cancels the reset of the respective alarm.
	Copy lines	Copies the selected alarms.
	Time base setup	Opens a dialog for setting the time zone for the time information shown in alarms.
	Selection display	Opens a dialog for filtering alarms. Define your own filter criteria or change or remove filters defined in the engineering system.
	Sorting setup	Opens a dialog for setting custom sorting criteria for displayed alarms.
	Display options setup	Opens a dialog in which you set which alarms the currently displayed alarm list displays.
	Configuration of the disabled alarms	Opens a dialog for configuring the display options of the disabled alarms.
	Export	Starts exporting the alarms to a CSV file.
	Select context	For context-based filtering of alarms. The alarm control only shows alarms that fall within the time period of the selected context entry.

Operate alarm control

Note

Displayed alarms

The alarms that you see in the alarm control depend on which alarm view or alarm list you have selected in the toolbar.

Operation using the mouse

Selecting and operating alarms

- Click on an alarm.
- Click a button in the toolbar.

The function of the button is applied to the alarm.

Rearranging columns

You can change the column arrangement configured in the engineering here. See section Rearranging columns at runtime (Page 7451).

Sorting alarms by column

You can sort the alarms by column. See section Sorting alarms (Page 7292).

Operation using the keyboard

Press <Shift + Enter> until the focus is on the alarm control. Then select the alarm to be edited and operate it using the toolbar.

Use the following buttons for this:

Buttons	Description
<PgUp>	Selects the previous alarm.
<PgDn>	Selects the next alarm.
<Ctrl + Up> or <Home>	If multiple rows were selected, the first row of the selection is selected.
<Ctrl + Down> or <End>	If multiple rows were selected, the last row of the selection is selected.
<Ctrl + Left>	If multiple columns were selected, the first column of the selection is selected.
<Ctrl + Right>	If multiple columns were selected, the last column of the selection is selected.
<Tab>	Selects the next button in the toolbar.
<Shift + Tab>	Selects the previous button in the toolbar.
<Enter>	Executes the currently selected button.
<Shift + Page Up>	Scrolls to the left column-by-column.
<Shift + Page Down>	Scrolls to the right column-by-column.

Alternative operation

- Depending on the configuration, you can also operate the alarm control via the function keys.
- If the alarm control is configured accordingly, all information about the alarm is displayed in a pop-up for a selected alarm. To do this, the function "Show recent" (Autoscroll) must be switched off.

See also

Supported gestures (Page 7261)

Filtering alarms

Introduction

You can use filters to control which alarms you see in the alarm view in runtime. To do so, define filter conditions.

The following settings are available in the "Alarm filter" dialog:

Setting	Description
"AND/OR" column	Adds additional criteria to the existing criteria with the Boolean operations AND or OR.
"Criterion" column	Selection list with the available criteria. Criteria correspond to the alarm blocks in the alarm control.
"Operator" column	Selection list with the available relational operators.
"Setup" column	Free text field
"Remove" button	Removes the selected filter criterion.
"Up/down" button	Moves the selected filter criterion.
"Filter" area	Free text area for direct input and editing of filter criteria.

Requirement

The "Selection display" button is configured in the alarm control.

Procedure

The following example describes how to define a filter. In the example, a filter is defined that filters for alarms that contain the alarm text "Motor on" and have a priority less than or equal to 5:

1. Click the "Selection display" button.

   The "Alarm filter" dialog opens.
2. Click in the "Criterion" column and select the "Alarm text" entry.
3. Click in the "Operator" column and select the "Equal to" entry.
4. In the free text field of the "Setup" column, enter the value "Motor on".
5. In the next row in the "And / Or" column, click "Add" and select the AND logic operation.
6. Click in the "Criterion" column and select the "Priority" entry.
7. Click in the "Operator" column and select the entry "Less than or equal to".
8. Enter the value "5" in the free text field of the "Setting" column.

9. Confirm your entries.
10. Close the dialog.

With some alarm blocks, for example, you can define the start and end times or search texts for "Date" and "Time". Your input must be in the format required in the dialog.

Note

In multi-user systems, make sure that contents displayed in the "Alarm filter" dialog on a client have the same names on all servers.

When filtering by time, the start and stop values are not adjusted automatically when the time base of the alarm control is changed.

Example: At the PC location with the time zone "UTC + 1h", the alarm control has the "Local time zone" time base. You should filter by the time 10:00 to 11:00. Change the time base from "Local time zone" to "UTC". If you want to display the same alarms, change the filter to 9:00 to 10:00 hrs.

Result

The filter is applied to all alarm lists in the alarm view.

Time-based filtering

Define the filter period

During time-based filtering of the alarm control, always define two filter conditions linked via "And". For these conditions, use the operators "Greater than", "Greater than or equal to" and "Less than or equal to".

Do not use the "Equal to" operator. When filtering, you specify the filter period down to the millisecond. Internally, the time stamp of alarms is stored precisely down to the nanosecond and the missing information for nanoseconds is supplemented by 0. A search with "Equal to" will therefore only find alarms whose time stamp has the nanosecond value 0.

Examples:

You can use the following filter conditions to filter for alarms that were triggered between 12:00 and 12:01:

- Filter condition 1: "Raise time", "Greater than or equal to", 12:00:00.000
- Filter condition 2: "And", "Raise time", "Less than or equal to", 12:01:00.001

You can use the following filter conditions to filter for alarms that were triggered at 12:00:00.000 hrs:

- Filter condition 1: "Raise time", "Greater than or equal to", 12:00:00.000
- Filter condition 2: "And", "Raise time", "Less than or equal to", 12:00:00.000

Change time base

If the time base of the alarm control is changed, the start value and stop value are not automatically adjusted when you filter by time.
Example: At the location of the PC with the time zone "UTC + 1h", the time base "Local time

zone" was selected for the alarm control. If you filter for the time 10:00 to 11:00 and then change the time base to "UTC", you need to change the start value and stop value of the filter to 9:00 and 10:00 to display the same alarms as before.

See also

Display alarms for plant objects (Page 7289)

Display alarms for plant objects

Introduction

In the case of the corresponding configuration, the alarm control shows the alarms of the plant objects that are configured in the plant hierarchy:

- Automatic display
  When the HMI device is assigned to a plant hierarchy or a plant object, and a plant overview and an alarm control are configured for the screen, the alarm control automatically shows the alarms of the plant object selected in the plant overview.
- Manual display through filters
  If no plant overview is configured in the screen, filter the alarm control to display alarms of a plant object.

The alarm control offers the following options for plant object alarms:

- Display the hierarchy path of the alarm source
- Filter the alarm control by plant objects
- Display alarm log of a plant object
- Context-dependent display of plant object alarms

General requirements

- The plant hierarchy has been created and a device assigned in the engineering system.
- An alarm control with the column "Area" has been configured in the screen of the assigned device.
- Runtime is active.

Filter alarm control by plant objects

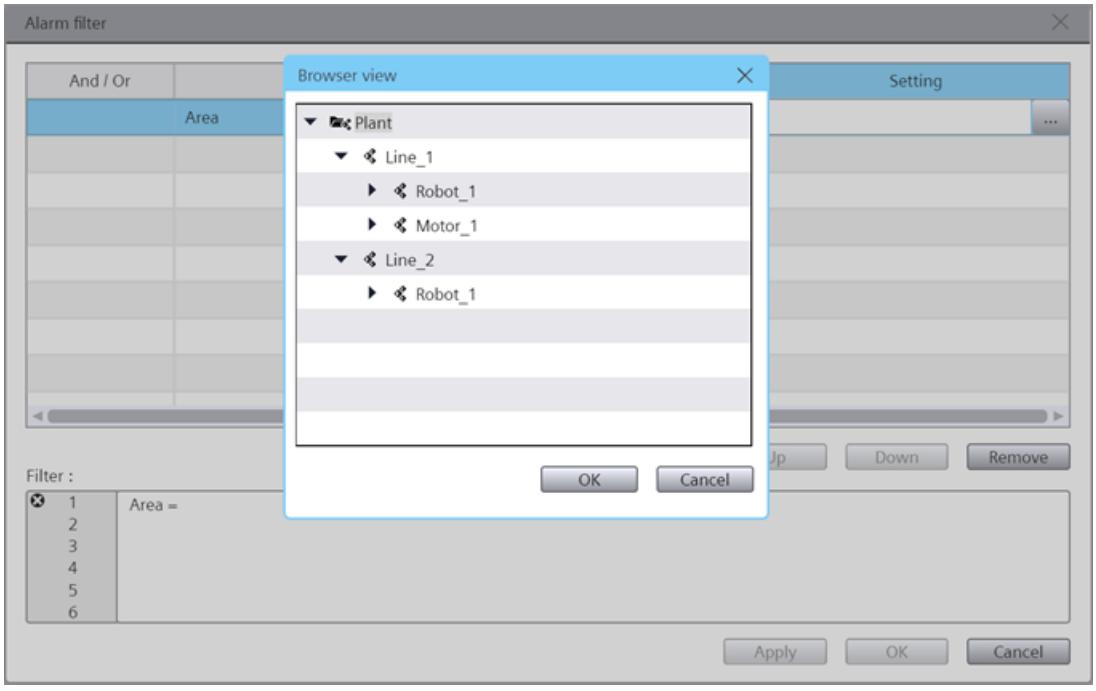
Additional requirements

- Alarms are available for a plant object from the plant hierarchy.

Procedure

1. In Runtime, click the "Selection display" button in the alarm control.
2. Select "Area" as the criterion in the "Alarm filter" dialog.
3. Click the cell of the "Setting" column

4. Click "...".
   A tree of the plant hierarchy is displayed.
5. Select a plant object and confirm your selection.



6. Under "Operand", select one of the following operators:
   - To display the alarms of the selected plant object, select "Same as".
   - To output the alarms of the lower-level plant objects, select "Begins with".

The alarm control shows its setting according to the alarms of the selected plant object or its lower-level plant objects. The "Area" column shows the complete path of the plant object.

Note

Display of the filter string for filters configured in engineering

The plant view is based on a type/instance architecture. When a filter has been configured in engineering that filters the alarm view by plant objects, you will first see a filter string with information from the type level in the "Filter" field of the "Alarm filter" dialog.

If you select an operand under "Operand" or a plant object under "Setting", the filter string changes to the instance level and adopts the device ID.

Display alarm log for a plant object

Additional requirements

- The alarm log contains entries for a plant object from the plant hierarchy.

Procedure

1. In runtime, click on the "Display logged alarms" button.

The alarm control shows the logged alarms of the plant object.

See also

Filtering alarms (Page 7287)

Plant overview (Page 7342)

Display context-dependent alarms of a plant object

This section describes how to show alarms that occurred on a plant object that you selected for a context that you selected.

Requirement

- An HMI device has been configured.
- An alarm control is configured in the device screen.
- The plant hierarchy has been created and assigned to the HMI device.
- There are alarms for the plant object.
- Contexts and context entries are available for the plant object.
- The "Select context" button is configured in the alarm control.

Procedure

1. In the alarm control, click the "Select context" button.
   The "Alarm context" dialog opens.
2. Click "..." and select the plant object whose data you want to display in the alarm control.
3. Select one of the contexts assigned to the plant object in the "Context" drop-down list.
   A list of the entries logged for the context appears under "Logged context values".
4. Select an entry.
5. Click "OK".

The alarm control shows the alarms of the plant object that fall within the time period of the selected entry. The information bar shows the values of the selected context.

Note

"AND" link with other filters

When a filter is defined for the alarm control, the filter condition and the context conditions are linked via "AND".

When no alarms appear in the alarm control, check your filter settings by clicking "Selection display".

See also

Contexts (Page 7240)

Sorting alarms

Introduction

You can control the columns according to which the alarm control sorts the alarms in runtime.

Examples for sorting alarms:

- In descending order by date, time, and alarm number. The latest alarm appears at the top.
- By priority
  You must have defined the priority of the alarms in the "HMI alarms" editor and configured the "Priority" alarm block in the alarm control. As a result, in a single-line alarm control, only the top-priority alarm appears in the alarm window. A lower-priority alarm is not displayed, even if it is more recent. The alarms are displayed in chronological order.
- The "Alarm state" alarm block is sorted by the type of state and not by the configured status texts. For an ascending sort order, the following order is used:
  - Active
  - Inactive
  - Acknowledged
  - Disabled
  - Activated
  - Automatic acknowledgment
  - Emergency acknowledgment
  - Active/Inactive

When sorting the alarm control by columns, define the sort order over up to four columns. An arrow and a number are shown on the right in the column header. The arrow indicates the sort order (ascending or descending). The number beside the arrow indicates the sort order of the column headers.

Requirement

- "Allow sorting" is enabled for the respective columns in the configuration of the alarm control.
- The "Show recent" function is paused in the alarm control.

Procedure

To filter alarms in the alarm control by column, follow these steps:

1. Click the column header by which you want to sort the alarms first.
   The number "1" is displayed with an arrow pointing upwards for ascending sort order or an arrow pointing downwards for descending sort order.
2. Optional:
   - To reverse the sort order for this column, click the column header again.
   - To cancel the sorting for this column, click the column header a third time.
3. If you want to sort by several columns, click the column header in the required order.

Alternatively, click the "Sorting setup" button and configure the sorting in the "Sorting" dialog.

Disabling individual alarms

Note

No locking and unlocking of PLC alarms

Locking and unlocking of PLC alarms for an S7-1500 PLC is not supported.

Introduction

If you disable an alarm, the alarm is not checked to determine whether the alarm condition applies. The alarm is not logged.

Note

Disabled alarm

Disabled alarms are no longer disabled after a restart of Runtime. Only alarms that are disabled directly in the automation system via data blocks remain disabled (disabled at source).

Requirement

- The "Visibility" and "Allow operator control" settings have been enabled for the following buttons in the engineering:
  - "Disable alarm"
  - "Enable alarm"
- The user is authorized to disable and enable alarms.

  Note

  The "Disable alarms" and "Enable alarms" authorizations must be configured directly one below the other. This is necessary because the authorization level used automatically for the "Enable alarms" authorization is directly below the "Disable alarms" authorization.

- An alarm is displayed on the HMI device.

Disable alarm

1. Select one of the following alarm lists in the alarm control:
   - "Show logged alarms"
   - "Show and update logged alarms"
   - "Show defined alarms"
2. Select the alarm.
3. Click "Disable alarm".

Result

The alarm is removed from the "Show active alarms" alarm list. Its alarm condition is no longer checked.

The alarm is visible in the alarm lists for logged alarms and defined alarms and has the status "Removed".

Enable alarm

To enable a disabled alarm, follow these steps:

1. In the alarm control, select an alarm list for logged alarms or defined alarms.
2. Select the alarm in the alarm list.
3. Click "Enable alarm".

The alarm condition of the alarm is checked again.

Shelving alarms

Introduction

You shelve an alarm for a specific period of time, for example, to prevent that a conformity error alarm affects the efficiency of your system.

Shelving can be canceled at any time. To do so, you use the buttons "Shelve alarm" and "Unshelve alarm" in the alarm control in runtime.

Requirement

- The "Visibility" and "Allow operation" settings have been activated for the following buttons in the engineering system:
  - "Shelve alarm"
  - "Unshelve alarm"
  - "Show active alarms"
  - "Display options setup"
- To unshelve: An alarm is displayed on the HMI device.

Procedure

To shelve an alarm, follow these steps:

1. Select one of the following alarm lists in the alarm control:
   - "Show active alarms"
   - "Show logged alarms"
   - "Update and display logged alarms"
   - "Show defined alarms"
2. Select the alarm.
3. Click the "Shelve alarm" button.

Result

The alarm is shelved. Its status remains unchanged.

The shelving creates a log entry. Shelved alarms are still available and logged in the system.

Note

Display of shelved alarms in the alarm list "Show active alarms"

Whether shelved alarms are visible in the alarm list for active alarms depends on the settings in the alarm list.

By default, the alarm list for active alarms does not display any shelved alarms.

Display shelved alarms

To display the currently shelved alarms, follow these steps:

1. In the alarm control, select the "Show active alarms" alarm list.
2. Click the "Display options setup" button.
3. Activate the option for shelved alarms.

Unshelve an alarm

To unshelve an alarm, follow these steps:

1. Display the shelved alarms.
2. Select the alarm in the "Show active alarms" alarm list.
3. Click the "Unshelve alarm" button.
4. If required, hide the shelved alarms from the "Show active alarms" alarm list.

Unshelving is canceled. Canceling the unshelving creates a log entry.

Acknowledging

Acknowledging alarms

The number of alarms to be acknowledged is indicated by a counter at the "Single acknowledgment" button or, if the alarm control was configured accordingly in engineering, by the information bar.

Introduction

You can acknowledge alarms in runtime according to your project configuration settings. You acknowledge alarms as follows:

- In the alarm control with the buttons "Single acknowledgment" and "Group acknowledgment", and for alarms with dual-mode acknowledgment also with the button "Single confirm".
- With customized buttons

When an operator authorization is configured for the buttons, the alarms can only be acknowledged by authorized users.

Acknowledgment variants

You acknowledge individual alarms or multiple alarms together in Runtime. The following options are possible:

- Single acknowledgment
  Acknowledgment of an alarm using the "Single acknowledgment" button of the alarm control.
- Group acknowledgment
  Acknowledgment of all pending, visible alarms that require acknowledgment in the alarm control using the "Group acknowledgment" button in the alarm control.
- Dual-mode acknowledgment
  When an alarm requires dual-mode acknowledgment, you must acknowledge both the enabling and disabling of the alarm. Or you acknowledge the alarm and reset it with the "Single confirm" button in the alarm control. The alarm status changes from "Active/ Acknowledged" to "Inactive".

Requirement

- The "Visibility" and "Allow operator control" settings have been enabled in the engineering for the following buttons of the alarm control:

	Single acknowledgment
	Group acknowledgment
	Single confirm
	Show recent

- For the single acknowledgment: An alarm that requires acknowledgment is pending.
- For the group acknowledgment: Several alarms that require acknowledgment are pending. The alarms do not require single acknowledgment.

Acknowledge alarms individually

To acknowledge an alarm, follow these steps:

- Read the alarm texts of the pending alarm and perform corrective measures, if necessary.
- Pause "Show recent".
- Select the alarm.
- Click "Single acknowledgment" in the alarm control.

Result

The alarm status is set to "Acknowledged". When the trigger condition for an alarm no longer applies, the alarm status is set to "Inactive" and no longer displayed on the HMI device.

Acknowledging alarms collectively

For group acknowledgment of alarms, follow these steps:

1. Read the alarm texts of the pending alarms and perform corrective actions, if necessary.
2. In the alarm control, click "Group acknowledgment".

Result

All pending alarms with the following properties are acknowledged:

- Requires acknowledgment
- Does not require single acknowledgment
- Visible

When the trigger condition for an alarm no longer applies, the alarm status is set to "Inactive" and no longer displayed on the HMI device.

Logging alarms

Logging basics

Introduction

An alarm log documents the alarms that occurred in the monitored process. You can use alarm logging to analyze error states and to document the process. When you analyze the logged alarms, you can extract important business and technical information regarding the operating mode of the plant.

With the appropriate configuration in engineering, logging alarms are created in runtime. If an error or limit violation occurs, for example, an alarm is output in runtime.

The alarm events are saved in a log database and/or printed out. The alarms logged in the database can be output in runtime if required, for example, in an alarm control.

The logged alarms are stored in a circular log that consists of multiple single segments.

With the appropriate configuration of the HMI device and a PLC connected to it, the alarms of the connected PLC are logged as well and made available in all configured languages.

Operating principle

An alarm is only logged if logging has been configured for its alarm class. The alarm logs are automatically created by the system in runtime.

Each alarm event of an alarm that has occurred is logged, for example, the status change from "Active" to "Active, acknowledged".

Note

Alarm classes for pure logging

Alarms of the alarm classes "Information", "OperatorInputInformation" and "SystemInformation" are only used for logging. In runtime, they are only displayed in the alarm lists "Show logged alarms" and "Show and update logged alarms".

Content of the alarm log

The alarm logs are used to store all alarm data, including configuration data. You can read all properties of an alarm from the logs, for example, alarm class, time stamp and alarm texts.

A new log segment with the new configuration data is generated whenever you edit configuration data of an alarm. This function prevents any change from influencing alarms logged previously.

The possible number of logged alarms depends on the database used.

Note

The time stamp of a logged alarm is always specified in standard UTC format (Universal Time Coordinated).

Because the alarm configuration is language-specific, the logs contain a configuration data table for each language configured.

Storage location and storage media

Log data are stored in a database. You can further process the saved data in other programs for analysis purposes, for example.

Backup for log segments

Take backups of your log segments to ensure complete documentation of your process.

Note

Backups can only be created if the Microsoft SQL Server is used.

Note

Segments from logs for which a backup was created can be restored in runtime. To do so, open SIMATIC Runtime Manager on the HMI device. See also section Restoring and deleting log segments (Page 7499).

Display of logged data

You can view the logged data on the HMI device with the buttons "Show logged alarms" and "Show and update logged alarms".

No logging due to overload

When an alarm cannot be written to the log after the configured number of attempts and within the defined time interval, the alarm is lost. The memory state is set to "StorageSystemWriteDataLost" internally. This documents that the number of alarms in the queue exceeds the configured high limit. No more alarms can be written to the log.

The alarm "SystemOverloadAlarm" of the alarm class "ALCL@%SystemInformation" is triggered. It is displayed in the alarm control but not logged.

Possible reasons for the overload:

- There are more alarms in the queue than can be processed.
- The alarms in the queue cannot be processed due to additional error conditions or memory states, for example, because the storage space is used up (memory state "StorageSpaceExceeded").

See also

Alarm classes (Page 7275)

Connecting and disconnecting the alarm log backup

Introduction

When you want to access the data of an archived alarm log, connect the log backup to the project. You can configure an automatic connection or connect the alarm log to the project via a script. The logged alarms are displayed in the alarm control.

If you no longer want to access the backup of a log segment, disconnect the log backup from the project.

Requirement

The relevant backup files in "*. ldf" and "*.mdf" format are stored locally.

Display Time Range

Alarms are only displayed if you have configured the time range in the alarm control accordingly.

Example

You have configured the time range so that only the alarms of the past 24 hours are displayed. When you connect to a log backup containing alarms that are older than 24 hours, these alarms are not included in the alarm control.

Automatically connecting to an alarm log

To automatically connect to the alarm log backup, follow these steps:

1. Insert the log backup files in the "RuntimeProjectPath\ProjectName\CommonArchiving" folder.
2. In runtime, the alarm log is automatically connected to the project.

If signing is enabled, signed log backup files that are changed will not be connected automatically. A WinCC system alarm is generated and an entry is added to the Windows event log in the "Application" section.

Connecting to the alarm log using a script

Using the "AlarmLogs" VBS object, you can link the log backup files to the project using a script. The log segments are then copied with the "Restore" VBS method to the "Common Archiving" folder of the Runtime project.

Automatically disconnecting the alarm log

To automatically disconnect the alarm log backup from the project, follow these steps:

1. Go to the folder "RuntimeProjectPath\ProjectName\CommonArchiving".
2. Remove the log backup files from the folder.

Disconnecting from the alarm log using a script

Using the "AlarmLogs" VBS object, you can disconnect the log backup files from the project using a script. The log segments are then removed with the "Remove" VBS method from the "Common Archiving" folder of the Runtime project. For additional information, see the description of the "AlarmLogs" VBS object and the "Remove" VBS method.

Display logged alarms

Introduction

You can display the logged alarms with the buttons "Show logged alarms" and "Show and update logged alarms".

Requirements

- An alarm log is configured.
- All logged data that is to be displayed must be stored locally on the logging server. The SQL server does not allow access to backup files held elsewhere, such as another computer on the network.
- The buttons "Show logged alarms" and "Show and update logged alarms" are configured in the alarm control.

Procedure

1. To display only logged alarms, click the "Show logged alarms" button in the alarm control:

   The alarm control shows the logged alarms. The display is not updated immediately when new incoming alarms occur.
   Each page shows a maximum of 1000 alarms. Use the "Previous page" and "Next page" buttons to change pages.

2. Click the "Show and update logged alarms" button in the alarm control to display logged and current alarms:

   The alarm control shows the logged alarms. The display is updated immediately when new active alarms occur.
   The alarm control shows a maximum of 100 alarms.

Restriction for the alarm list "Show logged alarms"

For log alarms with identical time stamp, it is possible in rare cases that log alarms are skipped when paging forwards and backwards.

To display the skipped alarms, page again, this time in the opposite direction.

Note

In the case of more than 1000 log alarms with identical time stamp, not all skipped alarms can be displayed by scrolling in the opposite direction.

Example

- The alarm log contains several 1000 log alarms. Ten alarms of the log have an identical time stamp. The first five are shown at the end of the current page.
  The alarm control is sorted by time stamp in ascending order.
- Click "Next page".
  You see the next 1000 alarms whose time stamp is higher than the time stamp of the last alarm shown on the previous page.
  The remaining five alarms with identical time stamp are skipped on the page change.
- Click "Previous page".
  You will see all ten alarms with identical time stamp as well as the next 990 alarms with lower time stamp.

Displaying alarm statistics

Introduction

The alarm statistics represent statistical calculations of logged alarms.

You can use a button in the alarm control to export the alarm statistics to an Excel file.

Note

Filter

A filter set in the alarm control is not effective in the alarm statistics.

Note

Display options

Display options selected via the "Configuration of the defined alarms" button in the alarm control are not effective in the alarm statistics.

Requirement

- Alarms are logged.
- For the following button of the alarm control, the "Visibility" and "Allow operator control" are enabled in the engineering system:

	Alarm statistics

Procedure

To display the alarm statistics in Runtime, proceed as follows:

1. Click the "Alarm statistics" button in the alarm control.

Result

The alarms to be displayed in the alarm statistics are specified in the engineering system. Depending on the engineering system, the following columns are displayed:

Column	Description
Number	Configured number of the alarm.
Frequency	Frequency of an alarm. The system counts the number of occurrences of an alarm with "active" status in the log. If the alarm number is not found, this alarm number is not included in the statistics.
Sum active active	Total display time of an alarm in seconds. The time period between the alarm states "active" and "active" is calculated.
Sum active inactive	Total display time of an alarm in seconds. The time period between the alarm states "active" and "inactive" is calculated.
Sum active acknowledged	Total display time of an alarm in seconds. The time period between the alarm states "active" and "acknowledged" is calculated.
Average active active	Average display time of an alarm in seconds. The time period between the alarm states "active" and "active" is calculated.
Average active inactive	Average display time of an alarm in seconds. The time period between the alarm states "active" and "inactive" is calculated.
Average active acknowledged	Average display time of an alarm in seconds. The time period between the alarm states "active" and "acknowledged" is calculated.

The calculation of the time of acknowledgment includes the "acknowledged" alarm state. This "acknowledged" alarm state includes the acknowledgment by the controller.

Note

For the calculation, alarms with the status "acknowledged" and "inactive" are only used if a suitable alarm with the status "active" is found in the result set beforehand.

If an alarm from the controller is pending and runtime is disabled and enabled several times, the alarm is entered into the log several times with the state "active". The alarm is also included multiple times in the calculation.

Operating alarm statistics

Introduction

Using the statistics setup, you can change the settings for calculating the alarm statistics. The following settings are available:

Setting	Description
Time range start	• Now The current time is displayed as the start time of the calculation. • Fixed The start time of the calculation can be changed as required.
Start time	Start time for the calculation. If the "Now" option is selected under "Time range start", the start time cannot be changed.

Setting	Description
Time range base	Unit of time for the calculation. The following settings are available: • Undefined The default time unit "Minute" is used with this setting. • Millisecond • Second • Minute • Hour • Day • Month • Year
Time range factor	The time range factor depends on the "Time range base" setting. For example, if the number 4 is set for the time range factor and "Minutes" is set for the time range base, all alarms that are logged within this period will be evaluated.

Requirement

- Alarms are located in the alarm log.
- For the following button of the alarm control, the "Visibility" and "Allow operator control" are enabled in the engineering:

	Statistics setup

- The alarm statistics are selected in the alarm control.

Procedure

To display the statistics setup in Runtime, follow these steps:

1. Click the "Statistics setup" button in the alarm control.
   Setup opens.
2. Change the settings as required.
3. Click the "OK" button.

Result

The calculation of the alarm statistics is displayed according to the changed settings.

17.3.4.3 Displaying tags in Runtime

Outputting the tag values

Overview

With WinCC you can output tag values in the HMI screen with different screen objects and change them.

- The I/O field is used for the input and output of process values.
- Bars are used for graphic display of the process values in form of a scale.
- Sliders are used for the input and output of process values within a defined range.
- The gauge is used to display the process values in form of an analog gauge.

In runtime you can also output tag values as table or as trend. You can use either process values or logged values as source for the tag values.

- Use a trend for the graphic display of tag values. Trends allows you to display the change in motor temperature, for example.
- Use a table to compare tag values. In the table you can, for example, compare fill levels of supply tanks.

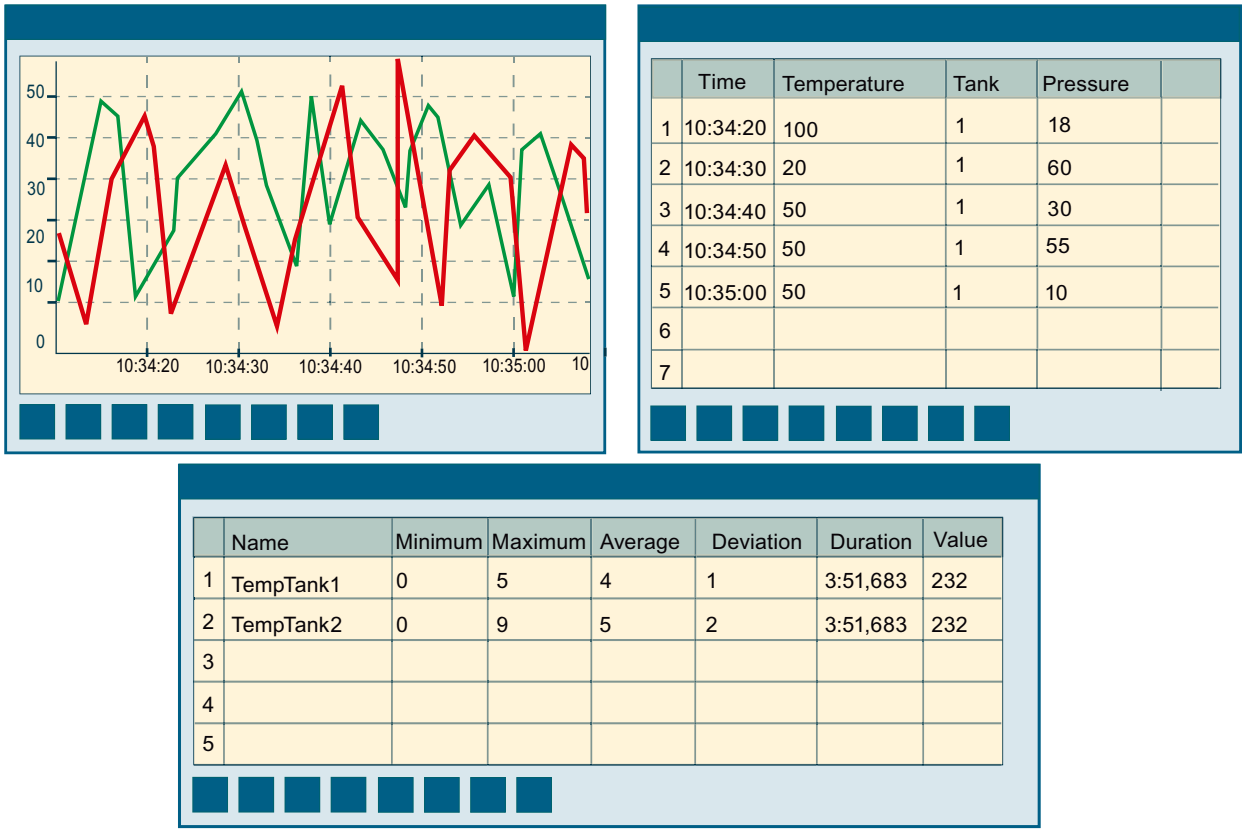
Controls for displaying tag values

To display tag values as a trend, use the trend control. The versions of trend views are available:

- "Trend control": You display a tag value over time, for example, the change in temperature. You can compare the current values and logged values or monitor the change in current values on the HMI device.
- "Function trend control": You display a tag value against a second tag value, for example, the engine speed against the heat produced.

You can use the "Trend companion" to create statistics, for example, from the displayed values. Furthermore, you can use the trend companion as reading assistance for the trend control.

To display tag values in a table, use the process control.



Displayed values

When configuring the trend control, specify which tag values are to be displayed.

- "Online": The trend is continued with current individual values from the PLC.
- "Log": In runtime, the trend control displays the values of a tag from a data log. The trend shows the logged values in a particular window in time. The operator can move the time window in runtime to view the desired information from the log.

Operating controls

Starting and Stopping Update

Introduction

You can continue the update of the data contained in the control with the "Start/Stop" buttons.

Some buttons stop the update automatically, e.g. "Define statistics area"

The appearance of the button indicates whether the update is stopped or not:

 The update has been stopped. To continue the update, click on the button.

 The update has been started. To stop the update, click on the button

Creating statistics for Runtime data

Introduction

You can generate an analysis of the process data for the Runtime data in the trend or process control. You can display the evaluated data in the trend companion.

Overview

Use the following buttons to create statistics of runtime data:

 "Start/stop"

 "Select time range"

 "Statistics area"

Requirement

- A trend control or process control is configured.
- A trend companion is configured and connected to the trend or process control.
- Runtime is enabled.

Displaying data in a statistics area window

Requirement:

The "Statistics area window" display mode is enabled in the trend companion.

To display data in the statistics area window of the trend companion, proceed as follows:

1. In the trend control or process control, click "Stop".
   The updated display is stopped, the process data continues being logged.
2. If you wish to evaluate data outside the displayed time range:
   - Click "Select time range".
     The "Time - Selection" dialog opens.
   - Enter the required time range.
     The data for the defined time range is displayed.

3. If you are using a trend control:
   - Click on the "Statistics area".
     In the trend control, two vertical lines are displayed on the right and left margin.
   - To define the statistics area, move the two lines to the desired position.
4. If you are using a process control:
   - Use the mouse to select the rows for the desired time range in the table.
     For different time columns with different time frames, you can select different time ranges for the calculation of statistics.
   - Click on "Statistic area" in the toolbar.

The evaluated data is displayed in the columns that you have configured in the statistics area window.

To continue with the display of Runtime data, click "Start".

Note

For additional statistical analysis of process data and logging of results, you can write the scripts yourself.

Displaying logged values

Introduction

Scroll through the displayed data of a log using the buttons in the toolbar of a trend or process control. If key combinations are configured, you can also use these for scrolling.

The buttons for browsing in logs are available only if data is supplied through logging tags.

The logged values of a tag are displayed within a time range in the trend or process control.

Overview

Use the following buttons to display logged values:


"First data record"


"Previous data record"


"Next data record"


"Last record"

Requirement

- Time range is configured.

Buttons for Archived Values

To scroll in archived values, proceed as follows:

1. To display the first data record of the time range, click on  .
2. To display the previous data record of the time range, click on  .
3. To display the next data record of the time range, click on  .
4. To display the last data record of the time range, click on  .

Elements of the information line

Elements of the information bar

The information bar of the trend control or process control can contain the following elements:

Icon	Name	Description
	Connection status[1]	No faulty data connections.
		Faulty data connections exist.
		All data connections are faulty.
"Line 1"[2]	Selected line	Shows the number of the selected line.
"Column 2"[2]	Selected column	Shows the number of the selected column.
"23.02.2010"	Date	Shows the system date.
"23:59:59"	Time	Shows the system time.
	Time base	Shows the time base used in the display of times.
[1]: If you double-click on the "Connection status" icon, the "Status of the data connections" window opens. The following properties of each data connection are listed in the window: • Name • Status • Tag name		
[2]: Only in the process control		

Basics of time range

The time range is the range from which the values at the HMI device are shown. The time range is determined by the start time and the end time. The time range is always in the past. If the end time is later than the current system time, the current system time is used as a temporary end time.

A distinction is made between the following time ranges:

- Static time range
- Dynamic time range

Static time range

The static time range is determined by fixed start and end times. The values are displayed within this time range.

Dynamic time range

The dynamic time range is determined by a period of time beginning with a fixed start time. The end time thus corresponds to the conclusion of the time period.

You set the time period as follows:

- Duration, e.g. 30 minutes
- The number of measurement points multiplied by the update cycle also produces a duration.

Configuring time range

Configure the time range for all controls. Configure the time range in the time column or in the time axis for the process control and the f(t) trend control. For the function trend control, configure the time range directly at the trend.

Exporting values

Requirement

- The "Export" button is configured in the control.

Procedure

1. Optional: For the export of a trend control, check the time format for the time axis configured in the control.
   The time axis of the export file takes on the time format configured in the control.
2. Click "Export" in the control.
3. Enter the name of the target file.

4. For the trend companion and process control: Choose whether all values are exported or just the values selected in the control.
5. Optional: Using "Select format", determine which separator and which character set the target file uses.

Note

Displaying Asian languages correctly in MS Excel

If Runtime is running in an Asian language, select the character set "UTF-8".

Trend companion

Trend companion basics

Function

The trend companion displays values or statistics from a control. The content of the trend companion is specified during its configuration.

Overview of the trend companion

The trend companion is connected to one of the following controls:

- Trend control
- Function trend control

In the trend companion, a "display mode" is specified during configuration. The display mode determines which data are shown in the trend companion.

Display mode

Three different display modes are available in the trend companion.

- Ruler window
  The ruler window shows the coordinate values of the trends on a ruler or values of a selected line in the table.
- Statistics area window
  The statistics area window shows the values of the low limit and high limit of the trends between two rulers or the selected area in the table. You can only connect the statistics area view to the trend control or the process control.
- Statistics window
  The statistics window displays the statistical evaluation of the trends. Among other things, the statistics include:
  - Minimum
  - Maximum
  - Average
  - Standard deviation
  - Integral

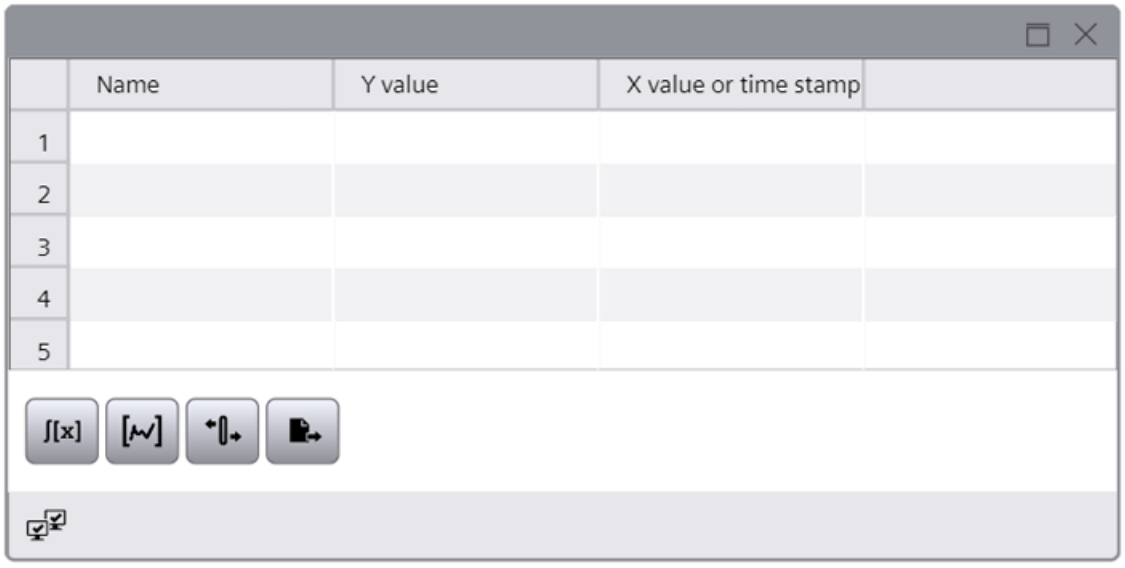
All windows can also display additional information on the connected trends or columns, such as time stamps.

Overview of trend companion

Note

With version V16, the "Trend companion" control is supported only for Unified PC. If you use the control under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel in which the control is configured must delete the control before compilation to version V16.

With the "Trend companion", you display evaluated data and statistics of a control in a table.



Buttons of the trend companion

The toolbar contains buttons for executing specific functions. Depending on the configuration, the following buttons are available for operator input:

Icon	Name	Function
	Statistical analysis	Displays the statistical values from a defined "statistics range" of the trend or process controls in the statistics window of the trend companion. Only available with a configured trend companion.
	Statistics area	Specifies the period for calculation of statistics.
	Ruler window	Displays a ruler that shows the coordinates of the intersection point with a trend in the trend companion. Requirement: The trend companion is configured with "Ruler window" display mode.
	Print	Reserved for future versions.
	Export	Exports all or selected data to a *.CSV file. Depending on the configuration and authorizations, the following options may be available: • Display export settings and start export • Select file name and directory

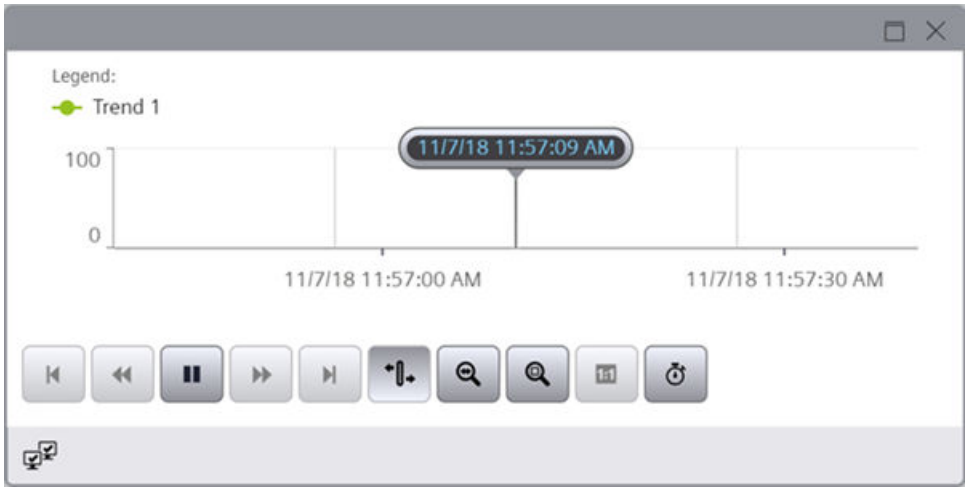
Rearranging columns

You can change the column arrangement configured in the engineering here. See section Rearranging columns at runtime (Page 7451).

Trend control

Overview of trend control

With the trend control, you show the currently pending process values or logged values as a trend over time. You design the trend display according to your wishes.



Note

Trend display in future time range

The trend area located in the future continues the last drawn value.

Buttons of the trend controls

The toolbar contains buttons for executing specific functions. Depending on the configuration, the following buttons are available for operator input:

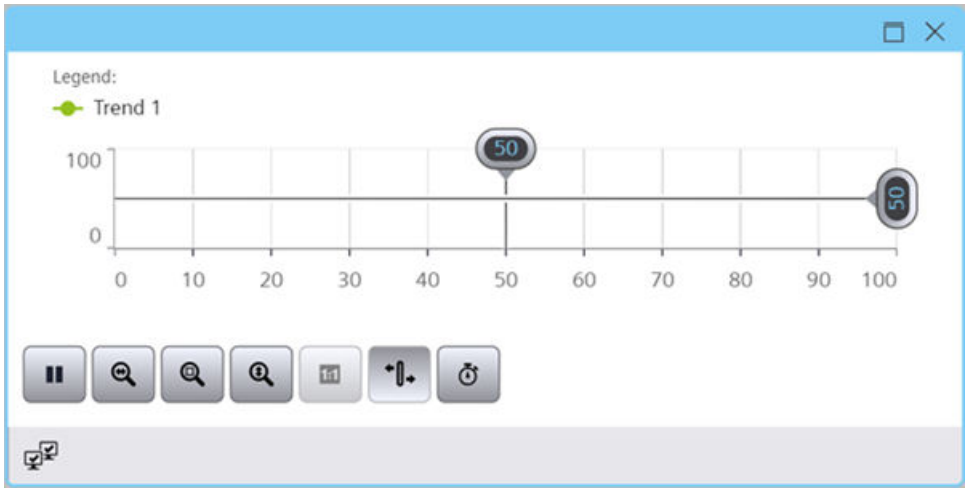
Icon	Name	Function
	First record	Shows the trend direction starting with the first logged value. Requirement: The values come from a process value log.
	Previous record	Shows the trend direction of the previous time range.
	Start/stop	Stops and starts the trend update. Started: The trend is continuously updated. It always shows the latest values. Stopped: New values are buffered and updated as soon as you start the trend update again.
	Next record	Shows the trend direction of the next time range.

Icon	Name	Function
	Last record	Shows the trend direction up to the last logged value. Requirement: The values come from a process value log.
	Previous trend	Displays the previous trend in the foreground.
	Next trend	Displays the next trend in the foreground.
	Ruler	Displays a movable ruler that shows the coordinates of the intersection point with a trend in the trend companion. With stopped trend update, the trend values are also displayed in tooltips. Requirement: The trend companion is configured with "Ruler window" display mode.
	Zoom time axis +/-	Zooms into or out of the time axis in the trend control. Left-click: Zoom in
	Zoom value axis +/-	Zooms in or out of the value axis in the trend control.
	Zoom area	Zooms in on the section of the trend control. You define the section by dragging with the mouse. Use the "Original view" button to return to the original view.
	Zoom +/-	Enlarges or reduces the view in the trend window.
	Move trend area	Moves the display in the trend area.
	Move axes area	Moves the display in the axes area.
	Original view	Returns to the original view from the zoomed display.
	Select time range	Opens a dialog in which you configure the time range.
	Select trends	Opens a dialog in which you set the visibility and sorting of trends.
	Select data connection	Opens a dialog in which you select the data source: • Process value log • Tag • Recipe (only function trend control)
	Statistics area	Enables you to define a time range for which statistical values are determined. Vertical lines which you use to set the time range are displayed in the trend window.

Icon	Name	Function
	Statistical analysis	Opens a statistics window to display the minimum, maximum, average, and standard deviation for the selected time range and the selected trend.
	Print	Starts printing the trends shown in the trend window.
	Export	Opens the dialog for saving the trend data in CSV format. The time axis in the export file takes on the time format configured in the control. If necessary, change the configuration of the time format in the control before the export.
	Select context	Shows the value range of the resulting data for analysis purposes

Overview of function trend control

With the function trend control, you display active or logged process values as a function of another tag in a trend. You design the trend display according to your wishes.



Note

Trend display in future time range

The trend area located in the future continues the last drawn value.

Button of the function trend control

The toolbar contains buttons for executing specific functions. Depending on the configuration, the following buttons are available for operator input:

Icon	Name	Function
	Start/Stop	Stops and starts the trend update. Started: The trend is continuously updated. It always shows the latest values. Stopped: New values are buffered and updated as soon as you start the trend update again.
	Zoom X axis +/-	Zooms into or out of the time axis in the trend control. Left-click: Zoom in <Shift + Left-click>: Zoom out Use the "Original view" button to return to the original view.
	Zoom area	Zooms in on the section of the trend control. You define the section by dragging with the mouse. Use the "Original view" button to return to the original view.
	Zoom X axis plus minus	Zooms into or out of the time axis in the trend control. Left-click: Zoom in <Shift + Left-click>: Zoom out Use the "Original view" button to return to the original view.
	Zoom Y axis plus minus	Zooms in or out of the value axis in the trend control. Left-click: Zoom in <Shift + Left-click>: Zoom out Use the "Original view" button to return to the original view.
	Original view	Returns to the original view from the zoomed display.
	Previous trend	Displays the previous trend in the foreground.
	Next trend	Displays the next trend in the foreground.
	Ruler	Displays a ruler that shows the coordinates of the intersection point with a trend in the trend companion. Requirement: The trend companion is configured with "Ruler window" display mode.
	Move trend area	You can move the trends in the trend window along the X axis and the Y axis using the button. Values from the future trend area apply the last displayed value.
	Move axes area	You can move the trends in the trend window along the value axis using the button.
	Select time range	Opens a dialog in which you configure the time range.

Icon	Name	Function
	Select trends	Opens a dialog for setting the visibility of trends.
	Select data connection	Opens a dialog in which you select the data source: • Process value log • Tag • Recipe
	Print	Click this button to print the trend shown in the trend window. The print job used during printing is defined in the configuration dialog in the "General" tab.
	Export data	This button is used for exporting all or the selected runtime data to a csv file.

Value aggregation

Introduction

If the number of process values or archive values to be displayed for the selected time range in a trend control is larger than the number of pixels available for the trend, they will be aggregated.

Which values are aggregated to a trend value depends on the loading time of the trend control. For this reason, screen changes can result in a change of the trend line.

Avoid aggregation

To avoid that values are aggregated, select a shorter time range or enlarge the width of the trend control.

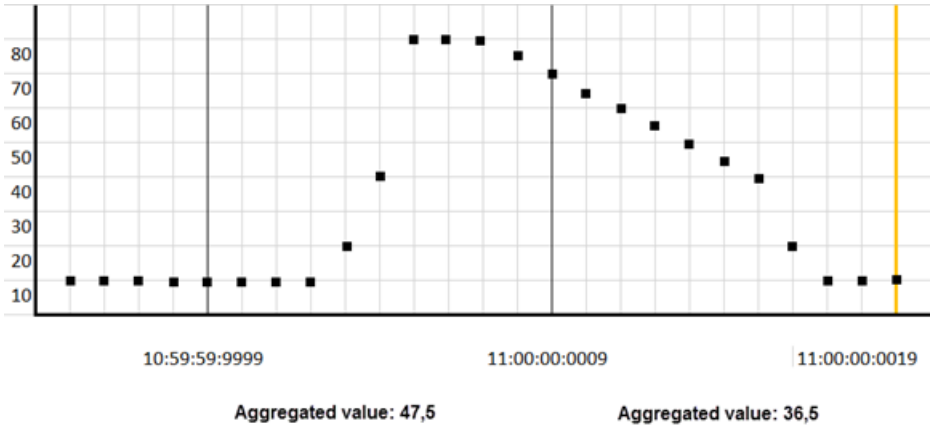
Example

- Pixels available for the trend: 600
- Measuring interval of the tag set as the data source: 10 times per s
- Time range: 10 minutes,

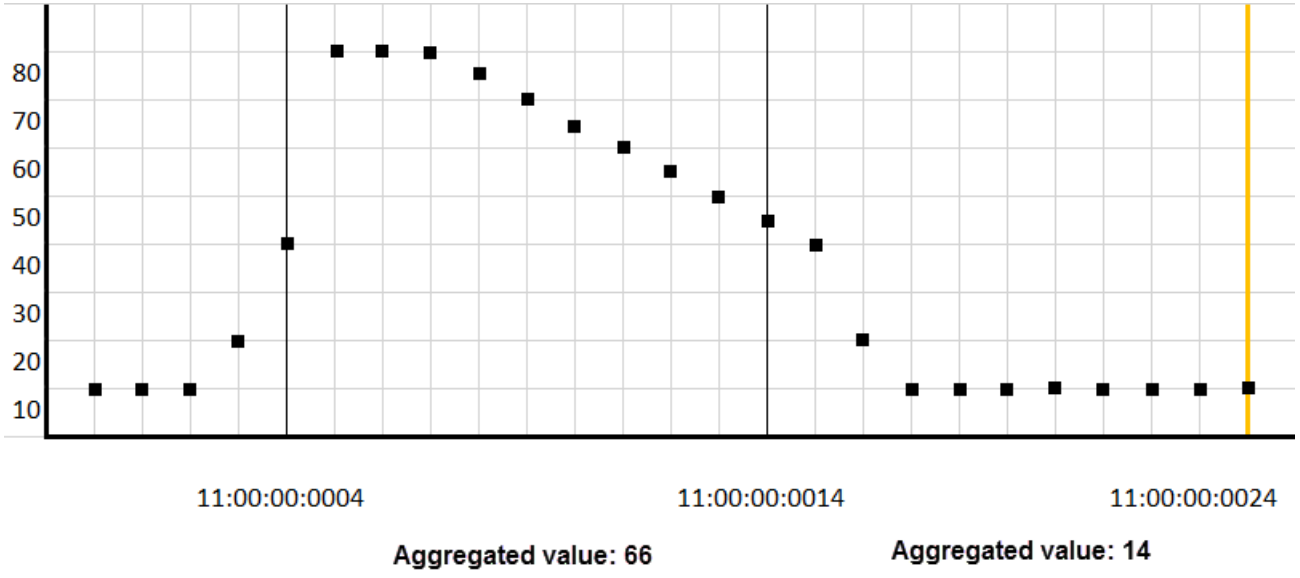
i.e. in the selected time range, 6000 values are measured or logged. When drawing the trend, 10 values are aggregated to each trend value.

The trend displays different values depending on the loading time. The following graphics illustrate how the last two aggregated trend values change when the loading time is 11:00:0019 instead of 11:00:0024.

- Loading time 11:00:0019:



- Loading time 11:00:0024:



Using the trend control

Online configuration of the trend control

Introduction

In Runtime, you configure online and thus change the appearance of the trend control.

During the configuration of the trend control, it is specified whether online configurations are retained or discarded during a screen change or after Runtime is ended.

Overview

Use the following buttons to configure the trend control in Runtime:

	"Select data connection"	Opens a dialog in which you can set the source from which a configured trend is supplied. Possible sources are the tags or logging tags of an HMI device or plant object and UDTs.
	"Select trends"	Opens a dialog in which you set the visibility and sorting of trends.
	"Select time range"	Opens a dialog in which you configure the time range.

See also

Select data connection of a trend (Page 7325)

Using the zoom functions in trend windows

Note

Scrolling in a zoomed in trend control

When the trend control is zoomed in, you can scroll using the mouse wheel:

- Move the mouse wheel to scroll up or down.
- Press <Shift> and move the mouse wheel to scroll to the left or right.

Introduction

Key functions can be used for zooming in on, zooming out of and returning to the original view for trends, axes and various zoom areas of the trend window.

Overview

The following zoom functions are available in the trend window:

	Zoom time axis +/-	Zooming in or out of time axis
	Zoom value axis +/-	Zooming in or out of value axis
	Zoom area	Zooming in on a trend control area
	Zoom +/-	Zooming in or out on trend
	Original view	Returning to the original view

Requirement

- The trend control is open
- Buttons with zoom functions are configured
- Runtime is enabled

Zooming in on a trend control area

Via the toolbar

1. Click "Zoom area" in the toolbar.
   The updated display is stopped.
2. Drag with the mouse to draw a box around the area to be zoomed.
   If there are at least two measured values within this area, the area of the trend is zoomed.
3. To return to the original view of the trend, click "Original view".
4. To restart the update, click "Start/Stop".

The default values are used for the axis.

Using the mouse wheel

Requirement: No zoom button was clicked in the toolbar.

1. Pause the update of the trend control.
2. Press <Ctrl> and move the mouse wheel.

Zooming in or out on trends

If you zoom in or out on a trend, the 50% value of the trend is always in the middle of the value axes.

Proceed as follows to zoom in or out on a trend:

1. Click "Zoom +/-".
   The updated display is stopped.
2. To zoom in on a trend, click on the trend with the left mouse button.
3. To zoom out on a trend, hold down the <Shift> key and click on the trend with the left mouse button.
4. To return to the original view of the trend, click "Original view".
5. To restart the update, click "Start/Stop".

The default values are used for the axis.

Note

If you change the value area of a value axis on the "Value Axis" tab in the configuration dialog while zooming, the visible zoom area is set to the new value area.

Zooming in on the time axis or value axis

While zooming with time or value axes, the 50% value of the trend is always in the middle of the axes.

Proceed as follows to zoom the time axis or value axis:

1. To zoom in or out on the time axis, click on "Zoom time axis +/-".
   The updated display is stopped.
2. To zoom in or out on the value axis, click on "Zoom value axis +/-".
   The updated display is stopped.
3. To zoom in on an axis, click on the trend control with the left mouse button.
4. To zoom out on an axis, hold down the <Shift> key and click on the trend control with the left mouse button.
5. To return to the original view of the trend, click "Original view".
6. To restart the update, click "Start/Stop".

The default values are used for the axis.

Zooming using touch gestures

Refer to the section Supported gestures (Page 7261).

Sorting trends

If a trend area contains multiple trends, you can select the order of the trends.

You have the following options:

- Specify the top trend
- Specify the order of all trends

Specify the trend order

Requirement

The "Select trend" button is configured in the toolbar.

Procedure

1. Click "Select trend" in the toolbar.
2. Click on a trend.
3. Move the trend to the selected position using the buttons.
4. Repeat these steps for the other trends.

Note

The trend at the top position is displayed in the trend area as the top trend.

Specify the top trend

In the drop-down list of the trend area, select the trend that you want to display as top trend.

Alternatively, use the "Select trend" button in the toolbar and move the desired trend to the top position.

Hiding and showing trends

Requirement

The "Select trend" button is configured in the toolbar.

Procedure

1. Click "Select trend" in the toolbar.
2. Disable the trend option to hide a trend.
3. Enable the trend option to show a trend.

Determining the coordinates of a point

Introduction

The "Ruler" button is used to determine the coordinates of a point on the trend by means of a ruler. You can zoom in on an area of the trend to make coordinate finding easier. If you display the ruler in the trend control, you can move the ruler at any time.

If you click on the trend with the mouse, several trend parameters are shown in the tooltip for the trend control.

Requirement

- A trend control is configured
- A trend companion is configured and connected with the trend control
- The "Ruler window" display mode is activated in the trend companion
- Runtime is activated

Procedure

Proceed as follows to determine the coordinates of a point:

1. Click "Ruler" in the trend control.
   The ruler is shown.
2. Move the ruler to the desired position with the mouse.
3. If you want to zoom in on an area, click on "Zoom area".
   - Move the ruler to the desired position with the mouse.
   - To return to the original view, click "Original view".

Result

In the ruler window of the trend companion, besides the X value/time stamp and the Y value, the data that you have configured in the trend companion is shown in the columns.

In the trend companion, the indices "i" and "u" can be displayed in addition to the values:

- "i.": The displayed value is an interpolated value.
- "u.": The displayed value has an uncertain status:
  - The start value after Runtime activation is unknown
  - A substitute value is used

  Note

  You can also display the "uncertain" status of a value in the displayed trend curve. You must activate the "Value with uncertain status" option on the "Trends" tab under "Limits".

Alternative procedure

Alternatively, you can also connect the trend companion to the process control. In the "ruler window" display mode, the values of the selected row are displayed in the trend companion.

Select data connection of a trend

You have the option to set in Runtime the source from which a trend is supplied.

Possible sources:

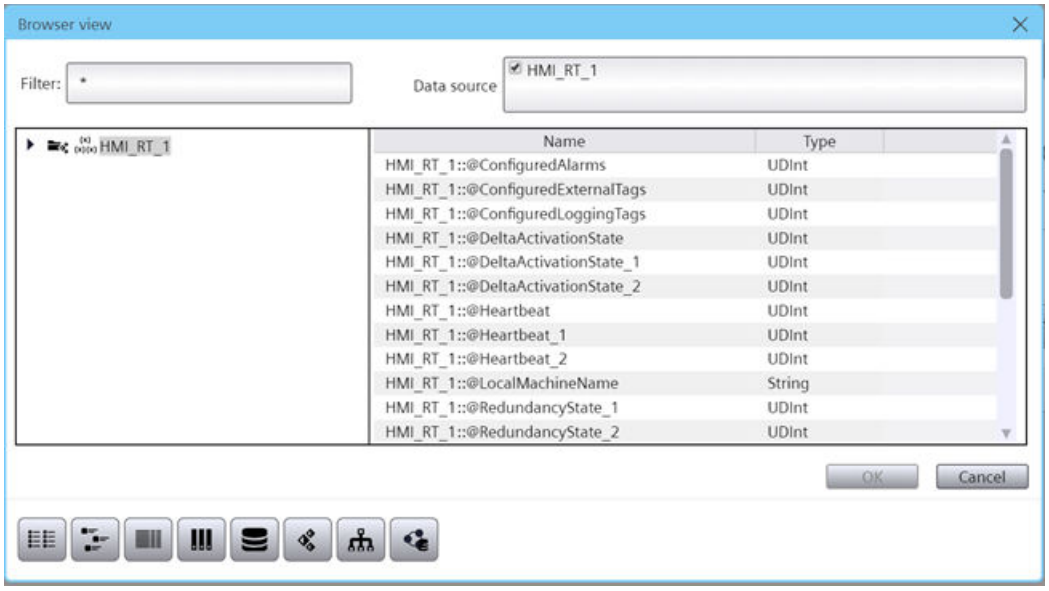
- Tags and logging tags of an HMI device, plant object or PLC
- UDTs

Requirement

- An HMI device has been configured.
- A trend control is configured in the screen of the device.
- To display logging tags: A data log has been configured.
- To display the tags of a plant object: The plant hierarchy has been created and assigned to the HMI device.
- Runtime is active.

Procedure

1. Click on "Select data connection" in the toolbar of the trend control.
   The "Selection of logs/tags" dialog opens.
2. Click "Trend:" and select a trend.
3. Click "Tag".
   The "Browser view" dialog opens in which you specify how the selected trend is supplied with data.



4. (Optional) Define in a filter.
5. Use the toolbar to configure the display in the dialog:

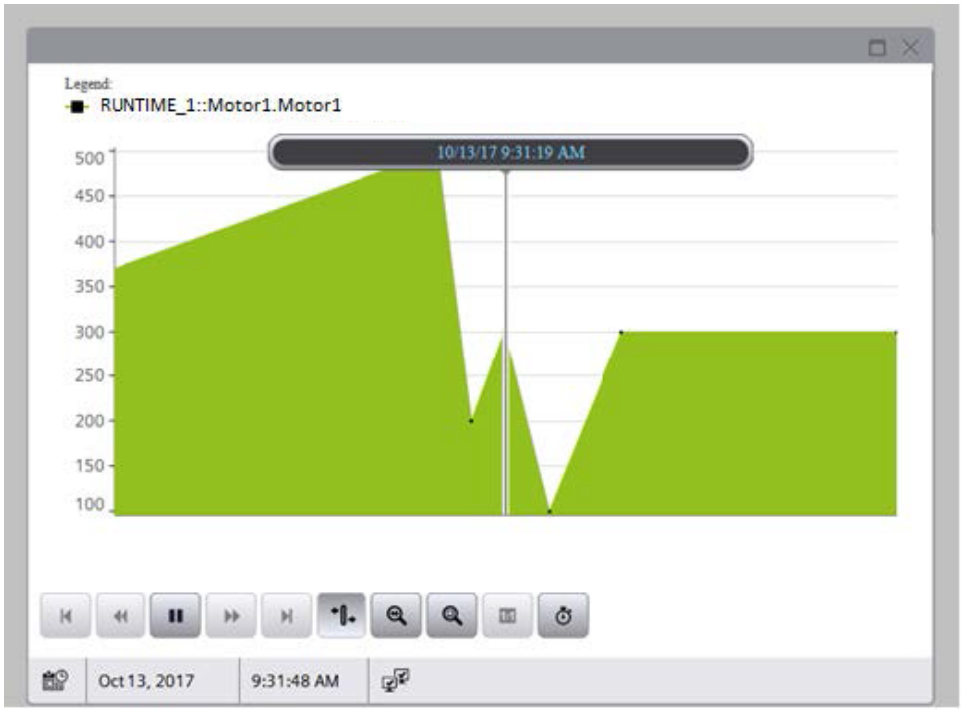
	"Small icons"
	"List"
	"Details"

6. Use the toolbar to configure the contents of the dialog:

	"Online tags"	Shows the device and its tags.
	"Logging tags"	Shows the device and its logging tags.
	"CPM"	Shows the plant hierarchy and the plant object tags.
	"UDT"	Shows the device and its UDTs.
	"CPM logging tags "	Shows the plant hierarchy and the logging tags of the plant objects.

7. In the tree, select the object whose data you want to display in the trend control.
8. Select a tag as the data source.
9. Confirm your entries.

The values of the tags are displayed in the trend control. If the path belongs to a plant object, the path of the plant object is also shown in the trend control.



Changing the time range of a trend

Procedure

To configure the time range, follow these steps:

1. Click "Select time range" in the toolbar of the trend control.
   The "Time selection" dialog opens.
2. Under "Time axes", select the time axis with the time range you want to adjust.
   Under "Trend area", you can see to which trend area the selected time axis belongs.
   If the trends in a trend control are to be displayed with a common time axis, the specified time range applies to all trends.
3. Configure the time range as described below.

Note

The format of date and time depends on the Runtime language used.

Configure time range

1. Select the "Time interval" entry in "Setting".
2. Select date and time of the start time.
3. Set the duration of the time range. To do this, enter a factor and select the time unit.
   Example: "90" as the factor and "Seconds" as the time unit for a duration of one and a half minutes.
4. Confirm your entries.

The time range of the time axis is adjusted:

- If the preset start time has been changed:
  - The trend update is paused.
  - The time axis starts with the selected start time.
- If the preset start time has been retained: The trend update continues. The preset start time is not included in the time axis.
- The duration of the time axis results from the factor and time unit.

Configure start time and end time

1. Select the "Start time and end time" entry in "Setting".
2. Select the date and time of the start time and end time.
3. Confirm your entries.

The time range of the time axis is adjusted:

- If the preset start time and/or end time has been changed:
  - The trend update is paused.
  - The time axis starts with the start time.
- If the preset start time and end time have been retained: The trend update continues. The preset start time and end time are not included in the time axis.
- The duration of the time axis results from the start time and end time.

Configure number of measuring points

1. Select the "Measuring points" entry in "Setting".
2. Select date and time of the start time.
3. Enter the number of desired measuring points under "Measuring points".
4. Confirm your entries.

The time range of the time axis is adjusted:

- If the preset start time has been changed:
  - The trend update is paused.
  - The time axis starts with the start time.
- If the preset start time has been retained: The trend update continues. The preset start time is not included in the time axis.
- The duration of the time axis results from the number of measuring points multiplied by the update cycle.

See also

Basics of time range (Page 7310)

Display context data of the plant objects in a trend control

This section describes how to show context-dependent data of a plant object in the trend control.

The evaluation is relevant, for example, in connection with the WinCC Performance Insight in order to analyze the effectiveness or the fault rate of the plant.

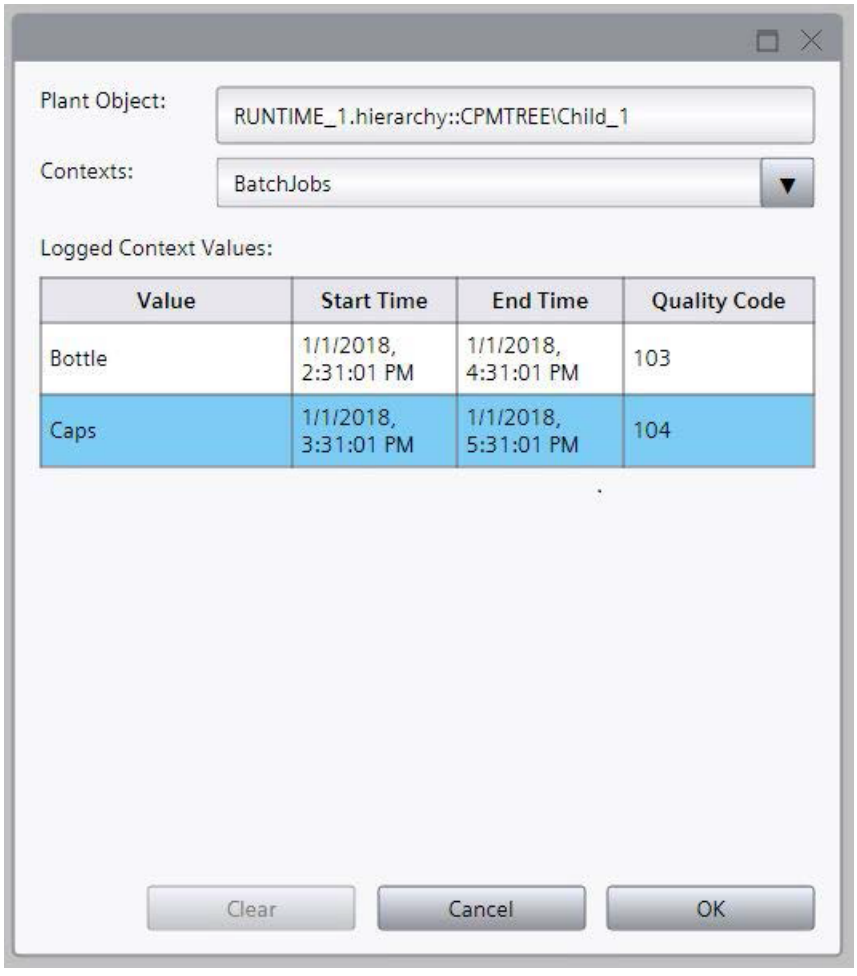
Requirement

- A trend control is configured in the screen of an HMI device.
- The plant hierarchy has been created and assigned to the HMI device.
- The data source of one of the trends in the trend control is a plant object.
- To display the logging tags of the plant object: A data log has been configured.

- Contexts are available for the plant object.
- The "Select context" button is configured for the trend control.

Procedure

1. In the trend control, click "Select context".
2. Select the plant object set as data source.
3. Select one of the contexts assigned to the plant object in the "Context" drop-down list.
   A list of the entries logged for the context appears under "Logged context values".
4. Select an entry.
5. Click "OK".



Result

The time period of the selected entry is applied to the time axis of the trend area. The trend represents the data that falls within the time period of the selected entry.

Note

Effect on other trend areas

If the plant object selected as data source has multiple interface tags and trends from other trend areas of the trend control display these tags, their time axes are also updated accordingly.

See also

Select data connection of a trend (Page 7325)

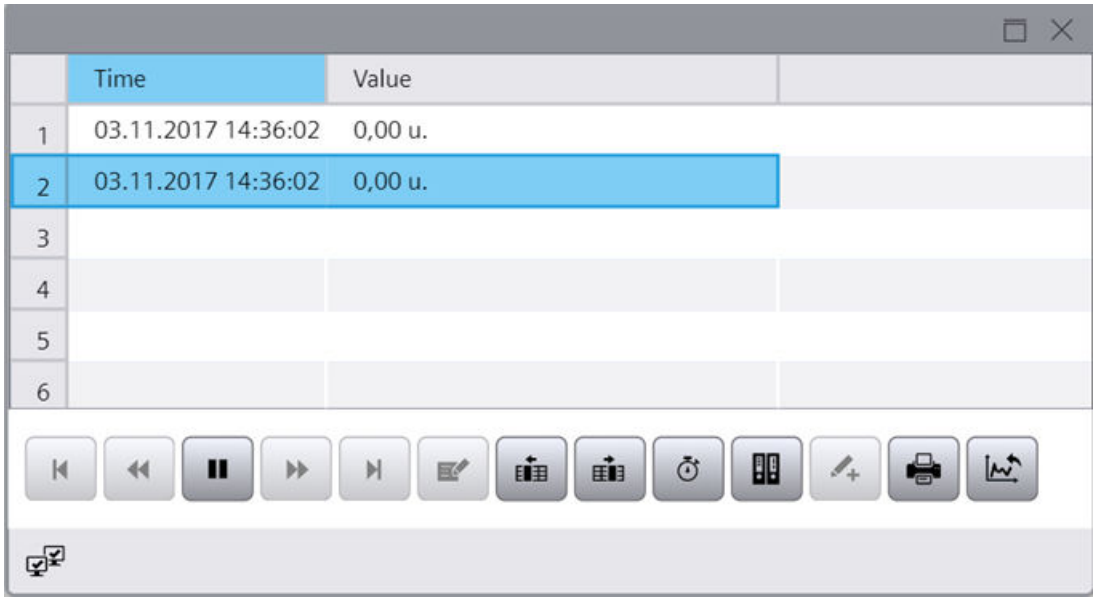
Contexts (Page 7240)

Process control

Overview of process control

With the process control, you display active or logged process values in a table. You design the display of the table as you wish.

You create statistics from selected data. You also export the data for further use.



Buttons of the process control

The following table shows the buttons that are available in the process control:

Icon	Name	Function	ID
	"First data record"	Displays the history of a tag within a specified time range starting with the first logged value. Requirement: Values come from a process value log.	0
	"Previous data record"	Displays the history of a tag within the previous time interval, based on the currently displayed time interval. Requirement: Values come from a process value log.	1
	"Start/stop"	Stops and starts the column update. The values are buffered and updated as soon as you start the column update again.	2
	"Next data record"	Displays the history of a tag within the next time interval, based on the currently displayed time interval. Requirement: Values come from a process value log.	3
	"Last record"	Displays the history of a tag within a specified time range ending with the last logged value. Requirement: Values come from a process value log.	4
	"Edit"	Activates editing of table entries. To edit a value, double-click in the desired table cell. Requirement: Updated display is stopped.	5
	"Previous column"	Moves the value column in front of the previous value column. The function refers to the value columns that are assigned to a time axis.	6
	"Next column"	Moves the value column to behind the next value column. The function refers to the value columns that are assigned to a time axis.	7
	"Select time range"	Opens a dialog in which you configure the time range.	8
	"Select data connection"	Opens the dialog for selecting the logs and tags of an HMI device, plant object or PLC that serve as data source for this table view.	9
	"Create archive value"	Creates a table entry for a log value. Enter the log value manually. Its time stamp corresponds to the time at which you added the table entry.	10
	"Delete archive value"	Deletes a logged value.	11
	"Export"	Exports all or selected data to a *.CSV file. Depending on the configuration and authorizations, the following options may be available: • Display export settings and start export • Select file name and directory	12

Using the process control

Online configuration of the process control

Introduction

In Runtime, you configure online and thus change the layout of the process control. The process control configuration specifies whether online configurations are retained or discarded on a screen change or after Runtime is ended.

Overview

The following buttons make online configuration possible in process control:

"Select data connection"

"Select time range"

Changing the data connection

The following table shows the configuration options for data connection:

Field	Description
Value column	Choose the configured value column for which you want to change the data connection.
Data Source	Define whether the selected value column is supplied with a logging tag or online tag.
Tag name	Select the tag name for the data connection.

Proceed as follows to change the data connection:

1. Click "Select data connection" in the toolbar.
   The "Log/tag selection" dialog is opened.
2. Choose the "Value column" for which you want to change the data connection.
3. Select the "Data supply" and the "Tag name".

Changing a time range

The following table shows the configuration options for the time range:

Field	Description
Time column	Select the configured time column for which you want to define a time range.
Time range	Set the time range: • If you want to define a fixed time interval, select the setting "Start to end time". Enter the date and time for each. • If you want to define a time period, select the setting "Time range". Define the date and time for the start time. The length of the time interval to be displayed is determined by multiplying the "Factor" by the "Unit of time". • If you want to display a certain number of values, select the setting "Number of measuring points". Define the date and time for the start time. Enter the desired number of measuring points in the input field.

To configure the time range, follow these steps:

1. Click "Select time range" in the toolbar of the process control.
   The "Time - Selection" dialog opens.
2. Choose the "Time column" for which you want to adapt the time range.
   If the columns of a process control are to be displayed with a common time axis, the specified time range applies to all columns.
3. Configure the time range.
   The entry format of the date and time depends on the Runtime language used.

Editing a table field

Introduction

You change the values displayed in the process control manually using the "Edit" button.

Overview

The following buttons allow you to edit the table fields:

"Start/stop"

"Edit"

Requirement

- The process control is configured
- The "Edit" button is configured
- Runtime is activated

Procedure

Proceed as follows to edit a table field in Runtime:

1. Click "Stop" in a process control.
   The updated display is stopped, the process data continues being logged.
2. Click "Edit".
3. Double click on the desired table field of a value column.
4. Enter the new value.
   The changed value is logged.
5. To continue with the display of Runtime data in the process control, click on "Start".

Moving value columns

You can rearrange the value columns assigned to a time axis.

Via the toolbar

1. Click on a column.
2. To move a column to the left, select "Previous column" in the toolbar
   The column is shifted one position to the left.
   If you have selected the first column, it is moved to the end of the value columns.
3. To move a column to the right, select "Next column" in the toolbar.
   The column is shifted one position to the right.
   If you selected the last column, it is moved to the beginning of the value columns.

With the mouse

See section Rearranging columns at runtime (Page 7451).

17.3.4.4 Screen window

Use

The "Screen window" object is used to display other screens of the project in the current screen. To continuously update the content of a screen window, for example, the object must be dynamized. Custom menus and toolbars can add specific buttons to the screen window.

You can also use independent screen windows independently of the screen in question. With appropriate hardware equipment and support by the operating system, you can also control multiple monitors and map processes in a more comprehensive and differentiated manner.

Layout

The settings for the position, geometry, style and color of the object are made during configuration.

In particular, the following properties are changed:

- Zoom factor: Defines the size of the embedded screen.
- Screen section: Defines the section of the embedded screen that is displayed in the screen window. If the embedded screen is larger than the screen window, you configure scroll bars for the screen window.
- Independent screen windows: Specified that the screen windows are displayed independently from the screen in which they were configured.

Note

Cascading screen windows

Screen windows can also display screens which, in turn, contain screen windows. Up to 14 cascaded screen windows can be displayed.

17.3.4.5 Web control

Introduction

The "Browser" control is designed for the visualization of simple HTML pages. It allows creation of centrally stored machine-specific descriptions, which are displayed from different HMI devices.

You have access to the data of the local user management in Runtime via a browser.

Note

Switching the functionality of the web control as a file explorer, in the following ways, for example, is not enabled in WinCC:

- Entry of a folder or drive, e.g. "\" or "C:", or
- Connection to an FTP server, for example, "ftp://"

One reason this function is not implemented is to prevent inadvertent changes to files, their deletion or execution.

When configuring, ensure that the operator can only enter valid Internet addresses, for example, by using symbolic I/O fields. Configure a password-protected input for service purposes.

Note

Page navigation in the web control

Whether you can navigate back and forth between the pages that you have viewed in the web control depends on the browser and browser versions in which Runtime is running. If the browser or browser versions do not support page navigation, the buttons in the web control are disabled.

Displayed content

Remember the following notes when using the control:

- The "Browser" control only shows contents that are supported by the browser in which Runtime is open.
- The control is implemented as IFrame. Pages with X-Frame option settings that prevent the display in an IFrame are not displayed in the control.
- As compared to a standard browser, the "Browser" control has limited functionality:
  - Navigation from the "Browser" control is not supported (top-level navigation).
  - Calls of queries and dialogs (pop-ups and modal dialogs) are only supported if they were activated in the file <Path for the WinCC Unified installation directory>WinCCUnified\WebRH\public\content\custom\CustomSettings.json:
    ```
    {"CustomSettings": {"HmiWebControl" : {"AllowPopups" :
    true,"AllowModals" : true}}}
    ```

    Note

    Pop-ups and modal dialogs stop the update.

  - Links to embedded files, for example, *.pdf or *.xls, are not supported.
  - Queries and dialogs that are conducted during the access of, for example, protected pages are not supported.

17.3.4.6 Media player

Use

In Runtime, the media player is used to play multimedia files via an https connection.

Layout

The settings for the position, style and color of the object are made during configuration.

In particular, the following properties are changed:

- Display operator controls: Specifies the buttons in runtime.
- Show tracker: establishes, whether a slider is available for the operation.

Supported file formats

The media player supports all formats that support the HTML5 video tag.

Restrictions

Note

Play restrictions

- The web control security settings do not allow local files to be played.
- Playing multimedia files in the Runtime control system depends on factors such as the installed operating system, the browser used and video and audio codecs installed on the machine.
  Examples:
  - Internet Explorer does not play any video file with an embedded .wav audio file.
  - Most browsers do not support .avi files.
- The browser determines which video formats are supported.
  You can find an overview of the video formats supported by popular browsers here (https://www.w3schools.com/html/html5_video.asp).
  You can find a detailed overview of the browser version used or between browsers here (https://html5test.com/compare/browser/index.html).
- iOS guidelines for the <video autoplay> element are available here (https://webkit.org/blog/6784/new-video-policies-for-ios/).

Note

Requirements for video files

To play video files in the Windows Server 2008 R2 SP1 and 2012 R2, install the Microsoft feature "Desktop Experience". You will find more detailed information on this topic on the Internet in the Microsoft documentation.

Note

Data loss when copying the project

If you copy the project to another PC, keep the following in mind:

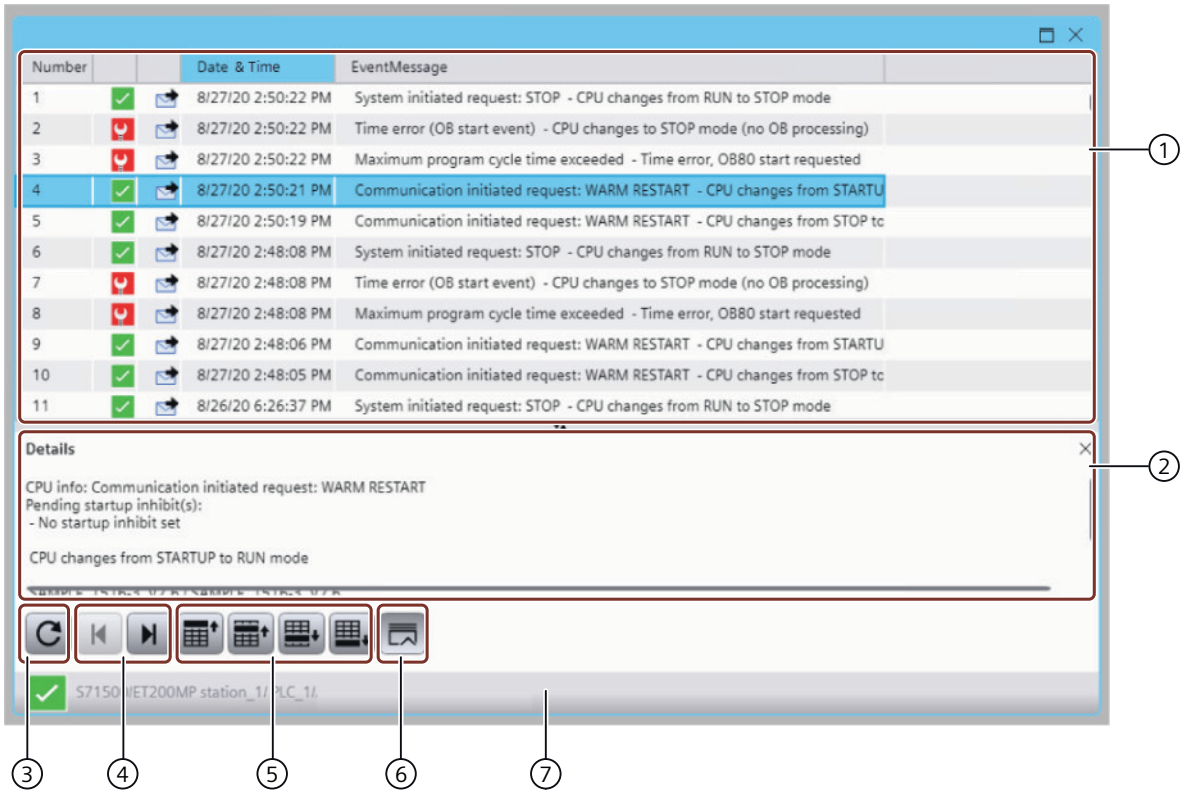
Files indicated in the WinCC Media Control are not copied along with the other files if they are dynamically linked and no UNC path is specified. You have to load the files into the project again.

17.3.4.7 System diagnostics view

The "System diagnostics control" object shows the diagnostic status of several PLCs via traffic light SVGs. The diagnostic status contains the overall status of all relevant PLCs. Navigation buttons can be used to navigate to the next PLC. The merged state is always the worst state of all PLCs.

Layout

In Runtime, the diagnostic messages of the selected PLC are displayed in the "System diagnostics control". The selected PLC can be changed using the buttons ④. Once Runtime has started, the events of the PLC with the most serious error are displayed.



① Grid view
② Detail view
③ Update the view of the diagnostics event
④ Switch to the next or previous PLC
⑤ Navigation buttons for the grid view:
jump to the first line
jump to the previous line
jump to the next line
jump to the last line
⑥ Enable/disable detail view
⑦ Information bar

The diagnostic buffer displays the diagnostics events of a PLC in a grid view ①. The grid view shows the last 200 diagnostics events of the PLC.

The first column shows the number of the entry.

The symbols in the second column indicate the event type of the PLC:

	Device in operation
	Maintenance required
	Maintenance necessary
	Error in the device

You can see the symbols of the incoming or outgoing status in the third column:

	Incoming event
	Outgoing event
	Incoming event for which there is no independent outgoing event
	User-defined diagnostics event

The fourth column shows the date and time of the event. You can see the event message in the last column.

Below the grid view, the detail view ② of the selected row from the grid view is displayed. You can enable or disable the detail view with the button ⑥.

When the screen is loaded, the "System diagnostics control" shows the PLC with the most severe error. If several PLCs are configured for system diagnostics, you can use the toolbar buttons ④ to switch to the next or previous PLC.

To update the "System diagnostics control", select the toolbar button ③. For performance reasons, no automatic update is performed.

At the bottom of the window, an information bar ⑦ is displayed with the diagnostic status and the name of the station/PLC.

Note

Rearranging columns

You can change the preconfigured column arrangement.

Languages in runtime

The alarms are displayed in the RT language selected by the user in the screen logon dialog. The Runtime language and the PLC language should be identical.

The PLC supports only three languages, which can be configured by the user in the engineering. If the PLC language and the Runtime language are different, the event text is displayed as follows according to the fallback mechanism:

- English US
- English UK
- the standard text "## text is missing ##"

See also

Rearranging columns at runtime (Page 7451)

17.3.4.8 Plant overview

Introduction

Note

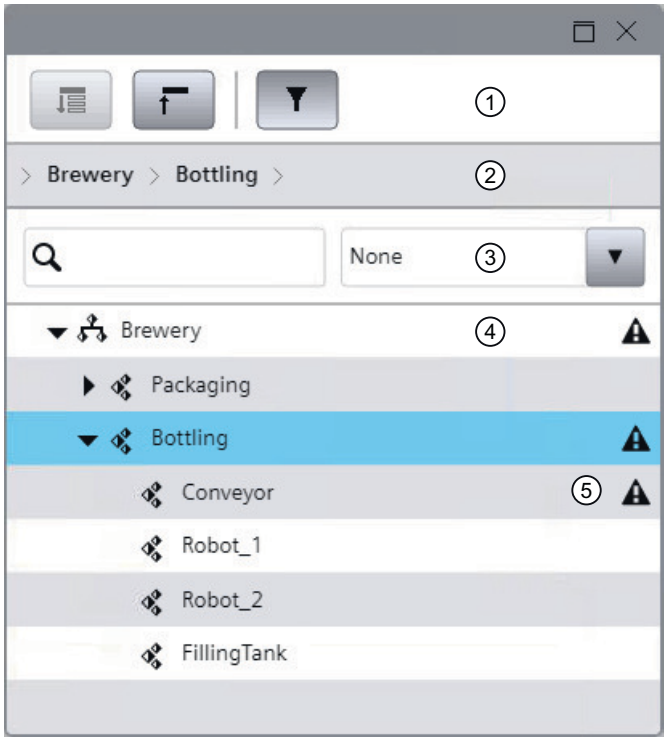
With version V17, the "Plant overview" control is supported only for Unified PC. If you use the control under Unified Comfort Panel, an error message of the compiler is returned. If the control is configured for the Unified Comfort Panels, it must be deleted before the compile.

The "Plant overview" object shows you the configured plant hierarchy in Runtime. In the plant overview, you can navigate through the system to the plant objects and see the plant at a glance.

With the corresponding configuration of the lower-level plant objects and the assigned HMI device during the engineering, the plant overview also offers you the following options:

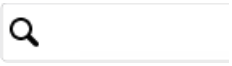
- Obtaining an overview of the plant objects for which alarms are available
- Displaying the alarms of a plant object
- Display of configured screens of a plant object.

Overview of the Plant overview



1 Toolbar
2 Menu bar
3 Filter bar
4 Plant tree
5 Alarm icon
Alarms are available for the plant object or one of its lower-level plant objects.

The following buttons are available in the toolbar and in the filter bar:

Icon	Name	Function
	Expand all	Expands all lower-level plant objects of the plant object selected in the control.
	Collapse all	Recursively collapses all lower-level plant objects of the selected plant object.
	Expand or collapse the filter bar	Opens or closes the filter bar.
		Filters the plant overview: • No filter: You see all plant objects • For plant objects for which alarms are available • According to plant objects for which screen windows are configured
	Search field	Filters according to the entered text.

When configuring in the engineering system, you can hide the toolbar and menu bar.

Requirement

- The plant view has been created and assigned to an HMI device.
- The "Plant overview" object is configured in the screen of the assigned HMI device.
- Optional:
  - The "Dynamic" navigation type is configured in the engineering system for the plant overview.
  - A root node is configured in the engineering system for the plant overview.
- Runtime is active.

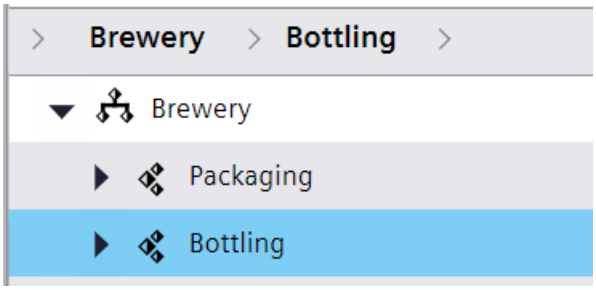
Operation

Expand and collapse plant tree

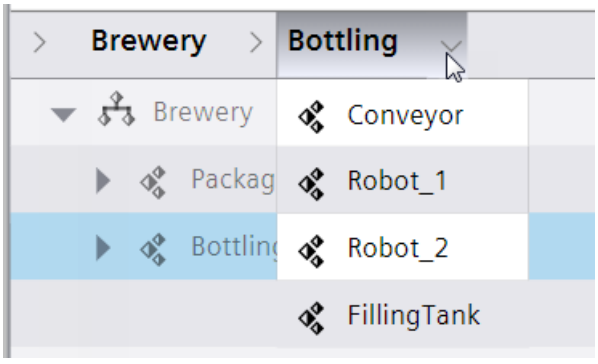
- To show all lower-level plant objects of a plant object, click the "Expand all" button.
  To collapse the plant tree, click "Collapse all".
- To expand only the lower-level objects of the next level, click the button with the triangle next to the plant object.
  To collapse the level again, click again on the button with the triangle.
  Alternatively, you can double-click the plant object to expand or collapse lower-level objects.

Select plant objects

- To select a plant object, click on the plant object in the plant tree.
  The path to the selected plant object appears in the menu bar of the "Plant overview" object:



- To see which lower-level objects a plant object displayed in the menu bar has at the next level, click the arrow in the menu bar next to the plant object.



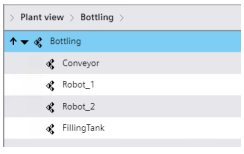
- To go from the menu bar to the overview, click on one of the plant objects shown in the menu bar.

Dynamic navigation

If dynamic navigation is enabled in the engineering system, specify the level from which the plant tree is displayed.

The buttons of the toolbar and the filter bar relate to the displayed area.

- To select a plant object, click on the object in the menu bar or double-click on the object in the plant tree.
  The levels below the selected plant object are available.
- To navigate up one level, click on the up arrow next to the plant object.



Root node

You have the option of defining a root node in the engineering system.

If a root node is configured, the root node and all objects below the root node are available in the plant overview.

See also

Display alarms for plant objects (Page 7289)

17.3.4.9 Plant overview with companion controls

Requirement

- The plant view has been created and assigned to a device.
- The "Plant overview" object is configured in the screen of the assigned device.
- The objects "Alarm control" and "Screen window" are configured in the screen of the assigned device and configured as companion controls of the plant overview.
- Screens are configured at the plant objects.
- Runtime is active.

Display alarms

To display the alarms of a plant object, click on the alarm icon.

The alarm control shows the alarms of the plant object.

Note

The alarm icon only appears when an alarm has occurred at the respective plant object or one of its lower-level objects. The alarm icon disappears again when the alarm is no longer present.

Show a screen of a plant object

To show the screen of a plant object, click on the plant object.

The screen window shows the screen of the assigned HMI device.

If you have not configured any screen window, a screen of the plant object with text box "$POName$" appears.

Note

"$POName$" is an expression with which the name of the plant object is resolved.

17.3.4.10 Parameter set control

Overview of parameter set control

Introduction

Note

With version V16, the "Parameter set control" is supported only for Unified PC. If you use the control under Unified Comfort Panel, an error message of the compiler is returned. Existing projects under Unified Comfort Panel in which the control is configured must delete the control before compilation to version V16.

Set up the machine for production in Runtime using parameter sets. The elements in a parameter set are defined in engineering by defining its parameter set type.

In Runtime, the parameter sets are displayed in the parameter set control. In the control, you manage the parameter sets and load a parameter set into the PLC to set up a machine for production.

Example

A bakery generates the following parameter set types in the engineering system:

- Bread
- Bread rolls
- Cake

The elements of the parameter set types define the ingredients of these products. For example, the parameter set type "Bread" has the following elements:

- Flour
- Salt
- Syrup
- Yeast
- Water

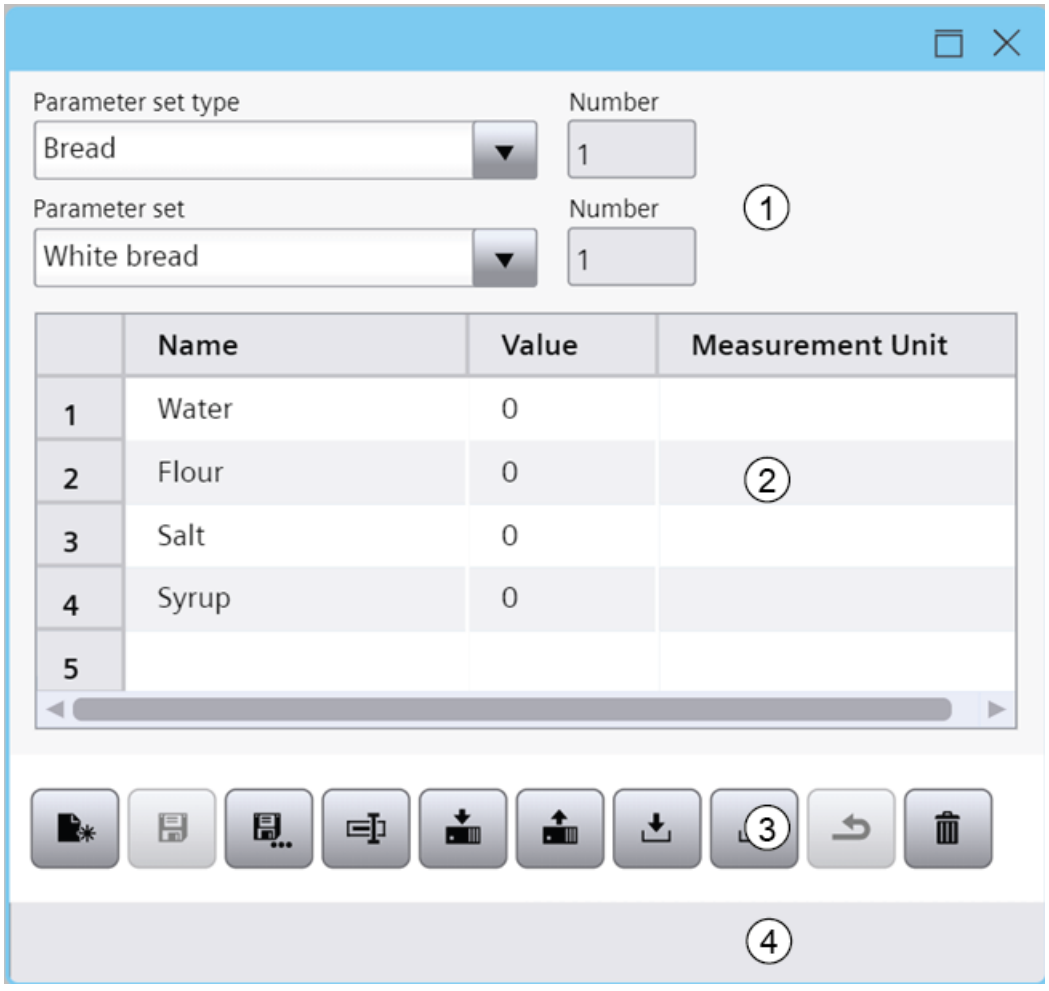
In Runtime, the bakery creates parameter sets for the "bread" parameter set type for the bread types to be produced:

- White bread
- Wholewheat bread
- French bread

The quantities required for this type of bread are entered in the elements.

During production, an operator selects the parameter set to be produced next and writes it to the PLC.

User interface



① Area for selecting the parameter set types and parameter sets

② Parameter table

Displays the values of the selected parameter set or parameter set type. The columns in the table depend on the configuration in engineering.

③ Toolbar

④ Information bar

The elements in the information bar depend on the configuration in engineering.

Note

Fixed parameter set type

The parameter set control in the engineering system can be configured so that you are only offered the parameter sets of a certain parameter set type and cannot select any other parameter set types.

Parameter set control buttons

The toolbar contains buttons for executing specific functions. Depending on the configuration, the following buttons are available for operator input:

	Button	Function
	Create	Creates a new parameter set.
	Save	Saves a parameter set.
	Save as	Opens the selection dialog for the storage path of the selected parameter set.
	Rename	Renames the selected parameter set. The new name must be unique.
	Write to PLC	Save the parameter set and writes it to the PLC.
	Read from PLC	Reads a Parameter set type or parameter set from the PLC.
	Import	Imports parameter sets to a CSV file.
	Export	Exports parameter sets to a CSV file.
	Cancel	Cancels the process.
	Delete	Deletes the selected parameter set. The table shows the default values at the parameter set type.

Rearranging columns

You can change the column arrangement configured in the engineering here. See section Rearranging columns at runtime (Page 7451).

Operate parameter set control

Create parameter sets

Requirements

- Parameter set types were configured in the engineering system.
- The parameter set control is configured in the screen of the device that is active in Runtime.

Create a new parameter set

To create a new parameter set, follow these steps:

1. Select a parameter set type in the parameter set control in "Parameter set type".
   The parameter table loads the columns and default values predefined at the parameter set type.
2. Click the "Create" button.

   Note

   Cancel creation

   Another parameter set parameters set type cannot be selected until you have saved the new parameter set or clicked on "Cancel".

3. Optional: Enter the name of the new parameter set in "Parameter set name".
   The name must be unique for the parameter set type.
4. Optional: Enter the ID of the parameters set in "Number".
   The number must be unique for the parameter set type.
5. The find the values of the parameters set by clicking in a table cell and modifying the value predefined by the parameter set type.
6. Confirm.

The parameter set is created and saved.

Create a version of the existing parameter set

To create a new parameter set based on an existing parameter set, follow these steps:

1. Select a parameter set type in the parameter set control in "Parameter set type".
2. Select a parameter set in "Parameter set".
   The parameter set table loads the columns and values defined for the parameter set.
3. Click the "Save as" button.
   The "Save parameter set as" dialog opens.
4. Optional:
   - Overwrite the automatically generated name in "Parameter set name".
     The name must be unique.
   - Overwrite the ID in "Number".

5. Confirm.
   The new parameter set is created.
6. To change the values taken over from the original parameter set, click in a table cell and enter a new value.
7. Click the "Save" button.

The parameter set is created and saved.

Edit parameter sets

Requirements

- Parameter set types were configured in the engineering system.
- The parameter set control is configured in the screen of the device that is active in Runtime.
- A parameter set was created in Runtime.

Procedure

1. In the parameter set control, select a parameter set type under "Parameter set type".
2. Select a parameter set in "Parameter set".
   The parameter set table loads the columns and values defined for the parameter set.
3. The new parameter set is created.
4. Click in a table cell and enter a new value.
5. Click the "Save" button.

Exchanging data with the PLC

Requirements

- Parameter set types are configured in the engineering system.
- The parameter set control is configured in the screen of the device that is active in Runtime.
- Write to the PLC: Parameter sets are available in Runtime.

Write parameter sets to the PLC

1. Select a parameter set and a parameter set type in the parameter set control.
2. Click "Write to PLC".

Result

- Unsaved data of the parameter set is saved.
- The parameter set is written to the PLC of the parameter set type.

Reading parameter sets or parameter set types from PLC

Reading a parameter set

1. To read a parameter set from a PLC, select the parameters set type of the parameter set and the parameter set.
2. Click "Read from PLC".
3. Select one of the following options:
   - Overwrite parameter set
   - Create new parameter set
     Define the name and the number.

Reading a parameter set type

1. To read a parameter set type from a PLC, select the parameter set type.
2. Click "Read from PLC".
3. A parameter set is created for the parameter set type during import. Define the name and the number of the parameter set.

Result

- The parameter set or the parameter set type and a parameter set are read in.
- The parameter table is updated.
- The parameter set that is created for the imported parameter set type has the default values defined in the type.

Note

A parameter set cannot be read from the PLC if minimum and/or maximum values are defined for a parameter set type element and the value in the parameter set to be transferred is outside this range. A message is output.

Importing and exporting parameter sets

Requirements

- Parameter set types are configured in the engineering system.
- The "Parameter set control" control is configured in the screen of the device that is active in Runtime.
- For the export: Parameter sets are available in Runtime.

Import

1. Click "Import" in the parameter set control.
2. In the dialog "Import - Parameter set", select a TSV file with parameter sets.

3. To overwrite parameter sets in the parameter set control that have the same IDs as the imported parameter sets, activate the "Overwrite" option.

   Note

   If you deactivate overwriting and if a parameter set with the same ID or the same parameter set name exists in the parameter set control, the import of parameter sets is not possible.

   Any added parameter sets whose IDs and parameter set names deviate from the existing parameter sets are imported regardless of the "Overwrite" option.

4. Select "Check checksum" when importing a parameter data record that was exported with the "Generate checksum" option.
5. Click "OK".

Result

- The parameter sets from the file is stored in the database.
- In selecting your parameter set type, you will be prompted to select under "Parameter set".
- If the currently loaded parameter set was part of the import, it will be updated accordingly in the parameter set control.

Export

1. Select a parameter set type in the parameter set control.
2. Click the "Export" button.
3. In the "Export - Parameter set" dialog, select a storage location for the file with parameter sets.
4. Set the formatting settings:
   - Select the file format.
   - Select the list separator.
   - Select the decimal character.
   - Select "Generate checksum" to export the parameter data record with a checksum. Parameter data records with a checksum cannot be imported if they have been manipulated in the meantime.
5. Click "OK".

Result

All parameters of the parameter set type are exported.

Updating the UDTs

Parameter set types are linked to UDTs. If the UDT of a parameter set type is replaced or edited in the engineering system, the derived parameter sets are updated accordingly in Runtime after the next compilation and loading:

- Replacement of the UDT
  The parameter sets created in Runtime are retained. You adopt the elements and default values of the new UDT.
- Assignment of another UDT version
  The parameter sets created in Runtime are retained. New elements are assigned default values, deleted elements are removed.

17.3.4.11 Reports

Basics

Reporting in Runtime

Note

Restriction for Unified Comfort Panel

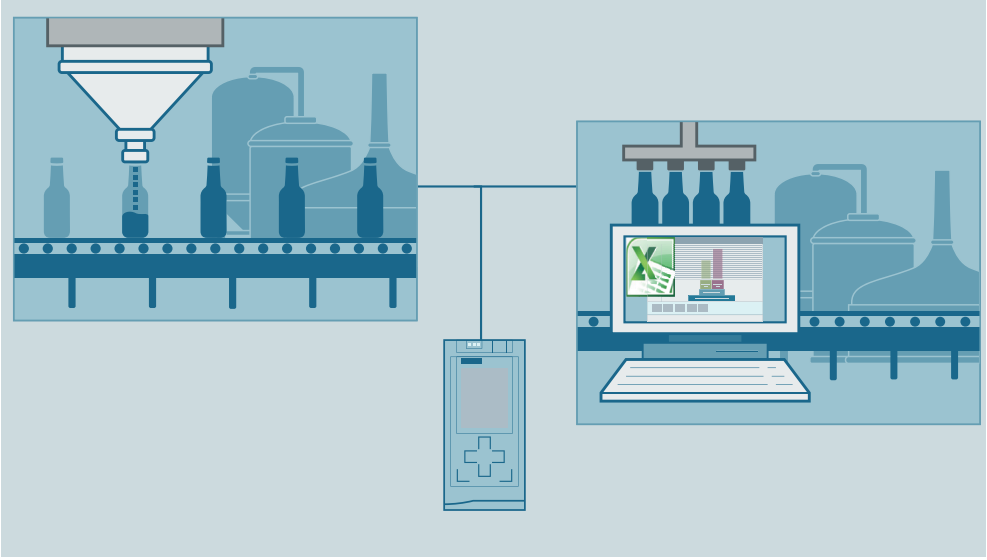
Contexts are not supported in V17 for Unified Comfort Panel. This option is not available in a report template with a Unified Comfort Panel as data source. When you generate a report on a Unified Comfort Panel whose report template uses this option, error entries are generated in the "ErrorLog" worksheet of the report.

Introduction

With WinCC Unified Reporting, you can generate tabular production reports (reports) in Runtime for the following project data:

- Logging tags and tags
- Log alarms
- Contexts:
  - User-defined contexts:
    These contexts are created and executed by a program created with the ODK API.
  - System-generated contexts
    When the Performance Insight and Calendar option packages are installed, these contexts are executed by the system during Runtime.
- Audit Trail of the Runtime device
- If Plant Intelligence options are installed, you can use the WinCC Unified Local Reporting option to generate production logs for additional project data.
  You can find more information in the Help for the respective Plant Intelligence option.

The production reports can be generated as XLSX file or PDF file and sent automatically as an email to a specified group of recipients. For example, you can generate an XLSX report that outputs all alarms occurring in a production line. You then distribute or archive the report for analysis purposes.



Functional scope

In the "Reports" control in Runtime, you configure report jobs that use the report templates defined in the Excel add-in. To do so, Reporting offers the following functions in Runtime:

- Maintenance of the global email settings (contact data and SMTP server configuration)
- Maintenance of job parameters, especially import and export of report templates
- Creating new report jobs and managing existing report jobs
- Overview of the generated reports
- Download or deletion of the reports

Basics of Reporting

Report templates

A *report template* is an Excel file (.xslx) that was created with the WinCC Unified Excel add-in. The report template has access to the data of the data source with which the add-in is connected.

For each report template, you define which segments are contained in the reports using the template and which data source items are evaluated by the segments.

After you have imported report templates into the "Reports" control in Runtime, you can select them for configuring report jobs.

Data sources

The *data source* is the source from which you select data source items when you configure the report template.

The following connection modes and data sources are available:

- Connection mode: Online
  The data source is the project that is running on the Runtime server to which the add-in is connected.
- Connection mode: Offline
  Data source is a configuration file. You generate the configuration file by exporting the data source items of the project to a file in the "Reports" control in Runtime. You can use this file to create additional report templates without connecting to a runtime server.

Options and data source items

Options control the types of data source items to which the report template has access.

Data source items are the specific objects whose data is read from the Runtime project during report generation.

The following options and types of data source items are available in Reporting, depending on the installed software:

Software	Option	Types of data source items
WinCC Unified basic installation	Alarms	Logging alarms Alarm statistics for logging alarms
WinCC Unified basic installation	Logging tag	Logging tags
WinCC Unified basic installation	Tag	Tags
WinCC Unified basic installation	User-defined column	User-defined texts or Excel formulas
WinCC Unified basic installation	Context	User-defined contexts Not available for Unified Comfort Panel
WinCC Unified basic installation	Audit	Audit
Performance Insight option package	Performance Insight	Local KPIs and operands of the PI option Performance Insight: • KPIs • Logged KPIs • Operands (counters and numerical operands) • Machine states • System-generated contexts
Line Coordination option package	Line Coordination	Jobs
Calendar option package	Context	System-generated contexts

Report jobs and job parameters

A *report job* is a job for generating reports in Runtime. A new report is generated each time the report job is performed.

The *job parameters* of the report order determine the details of its execution, such as which trigger it has, which report template it uses and the format of the report.

Report jobs are executed automatically when their trigger event occurs or manually by the user.

Reports

A *report* (production report) is an XLSX file or PDF file that is generated when a report job is executed in Runtime. The data source items from the Runtime project defined in the report template are read during generation, and their data are imported into a table in the report.

Using general Excel functions

In addition to the specific add-in functions, you also have access to the standard Excel functions in a report template. These include:

- Layout functions
- Functions for graphical preparation or analysis of the data imported from Runtime, such as charts, pivot tables and formulas

See also Tips on design and layout (Page 7449).

General requirements and restrictions

Installing the Excel add-in

The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

Update the operating system and the Excel version if necessary.

To install the add-in with a local Excel installation, MS Excel with build 16.0.6769 or higher is required.

Note

Note the Microsoft upgrade restrictions

If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

IIS settings for standalone installation of the Excel Add-In

To install the Excel Add-In on a PC without Unified Runtime, the same IIS (Internet Information Services) settings must be active in Windows that are required to install WinCC Unified Runtime on a PC.

You can find additional information in the "SIMATIC Unified PC Installation" user help section on the software and hardware requirements.

Unified Comfort Panel

The following restrictions apply to generating reports on Unified Comfort Panels:

Contexts	Contexts are not supported in V17 for Unified Comfort Panel. This option is not available in a report template with a Unified Comfort Panel as data source. When you generate a report on a Unified Comfort Panel whose report template uses this option, error entries are generated in the "ErrorLog" worksheet of the report.
Storage location of the Reporting database	The following folder on the SD card plugged into the panel is preconfigured as the storage location of the reporting database: • Device version V18: `media/simatic/X51/Reports` You can configure and load another storage location in TIA Portal in the Runtime settings of the Panel. • Device version <V18: `media/simatic/X51`
Storage location for reports	The "Reports" folder on the SD card inserted in the Panel is permanently pre-configured as the local main storage location for reports: `media/simatic/X51/Reports` For Panels with device version V18, you can configure and load a different local main storage location in the Runtime settings of the Panel in TIA Portal.

Enable Reporting

The use of Reporting requires that the Reporting functionality was enabled:

- For configuring report templates:
  Reporting must be enabled for the Runtime project that serves as the data source.
- For configuring report jobs and generating reports in Runtime as well as in a simulation:
  Reporting must be enabled for the Runtime project that is running on the HMI device or is being simulated.

The reporting functionality of a Runtime project is enabled in TIA Portal in the Runtime settings of its HMI device with the option "Enable reporting".

Note

Devices with a device version lower than V18

Reporting is always enabled for HMI devices with a device version lower than V18.

See also

Version compatibility (Page 7359)

Version compatibility

Introduction

When loading a Runtime project for which the "Reports" control has been configured, the general rules for version compatibility of WinCC Unified apply.

The rules described here also apply for the interaction between add-in, data source, report template and runtime version of the project in which reports are generated.

Compatibility between add-in and data source

The add-in can use the following data sources:

Add-in	Online data source	Offline data source
V16	Runtime project V16	Configuration file generated with a Runtime project V16
V17	Runtime project V16 or V17	Configuration file generated with a Runtime project V16 or V17

Compatibility between add-in and report template

The following report templates can be opened and edited in the add-in:

Add-in	Report template
V16	Created with a V16 add-in
V17	• Created with a V17 add-in • Created with a V16 add-in If the add-in is connected to a V17 data source when you open the report template, you will be prompted to migrate the report template to V17. If the add-in is connected to a V16 data source when the report template is opened, no migration is necessary.

Note

Migration of report templates

The migration of the report template is not reversible. A report template migrated from V16 to V17 can no longer be opened in a V16 add-in.

If migration is not desired, connect the add-in to a V16 data source before opening the report template.

Note

Scope of functions of report templates

The functions available in the configuration of the report template in the add-in depend on the version of the data source used by the add-in.

Compatibility between report template and runtime project

In a runtime project, reports can be generated using the following report templates:

Report template	Version of the runtime project
V16	V16 and V17
V17	V17

Workflow for working with reports in Runtime

Introduction

The following workflow describes which works are required in the "Reports" control so that production reports are generated in Runtime.

The reports can be stored as file in the file system and sent as an attachment to an e-mail. Alternatively, an e-mail without attachment can also be sent about the generation of the report. In this way, employees from management and production can be informed about the production situation promptly, regardless of their location.

You can send the e-mail using a secure SMTP server (authentication with user name and password or via certificate) or an unsecured SMTP server, for example, an internal company mail server.

Requirement

- Requirements in TIA Portal:
  - The necessary project data were configured for the HMI device for which reports are to be created.
  - The "Reports" control was placed on an HMI screen of the device.
  - The "Enable Reporting" option was enabled in the Runtime settings of the device.
  - (Optional) The storage locations for reports and the Reporting database were configured in the Runtime settings of the device.
- The HMI device has been compiled, uploaded to the Runtime server and its project is running.
- When using contexts: Contexts have been defined and executed in Runtime for the project.
- The Runtime server has access to report templates.
- For cross-project and cross-Runtime use of report templates: The data sources used in the report template can also be found on the HMI device. Make sure that the names and plant hierarchy are consistent.

Procedure

1. To send reports by e-mail, configure the global e-mail settings:
   - When one of the servers requires a certificate for sending e-mails, upload the certificate.
   - Create contacts for the e-mail receivers and e-mail senders.
   - Create the required SMTP server configurations.
2. Configure job parameters for report templates, triggers and targets.
   These job parameters will then be available to you for selection when configuring the report jobs.
3. Configure report jobs.
   Reports are generated in Runtime when the report jobs are executed.
4. (Optional) Perform report orders manually.
5. In the control, get an overview of which reports have been generated.
6. Download the reports, if necessary.
7. (Optional) To reuse the configuration of the "Reports" control, such as on a device in another network, transfer the existing configuration from the control from one device to the control of the other device.

Configuring job parameters

First, you configure which job parameters are available for selection during the configuration of the report jobs. You configure the following job parameters:

- The available report templates
  The report template defines which data the report outputs. Import and/or delete templates, if required.
- The available triggers
  The trigger defines when a report job is executed. Add triggers, edit triggers or delete them.
- The available targets
  Targets define whether reports are made available to users in the file system or via e-mails. Add targets, edit triggers, or delete them.

You set further job parameters while configuring a report job in the "Report jobs" tab.

Configuring a report job

You configure the following for each report job:

- Name of the report job
- Used report template

- Name of the reports generated by this template

  Note
  Texts through dynamic placeholders

  Placeholders are available to you when defining the report name. The placeholders are evaluated and replaced by text during execution of the report.

  See also Dynamic placeholder (Page 7386).

- Targets of the generated report
  To send e-mails, select a target of the type "E-mail".
- Per target: The target format of the generated report
  Possible formats: .XLSX and .PDF
- Trigger
- Comment
- Activate

See also

Setting global email settings (Page 7365)

Configuring job parameters (Page 7367)

Configuring report jobs (Page 7374)

Running a report job manually (Page 7382)

Downloading reports (Page 7382)

Transferring the control configuration (Page 7384)

Configuring report templates in the add-in (Page 7389)

The user interface of the "Reports" control

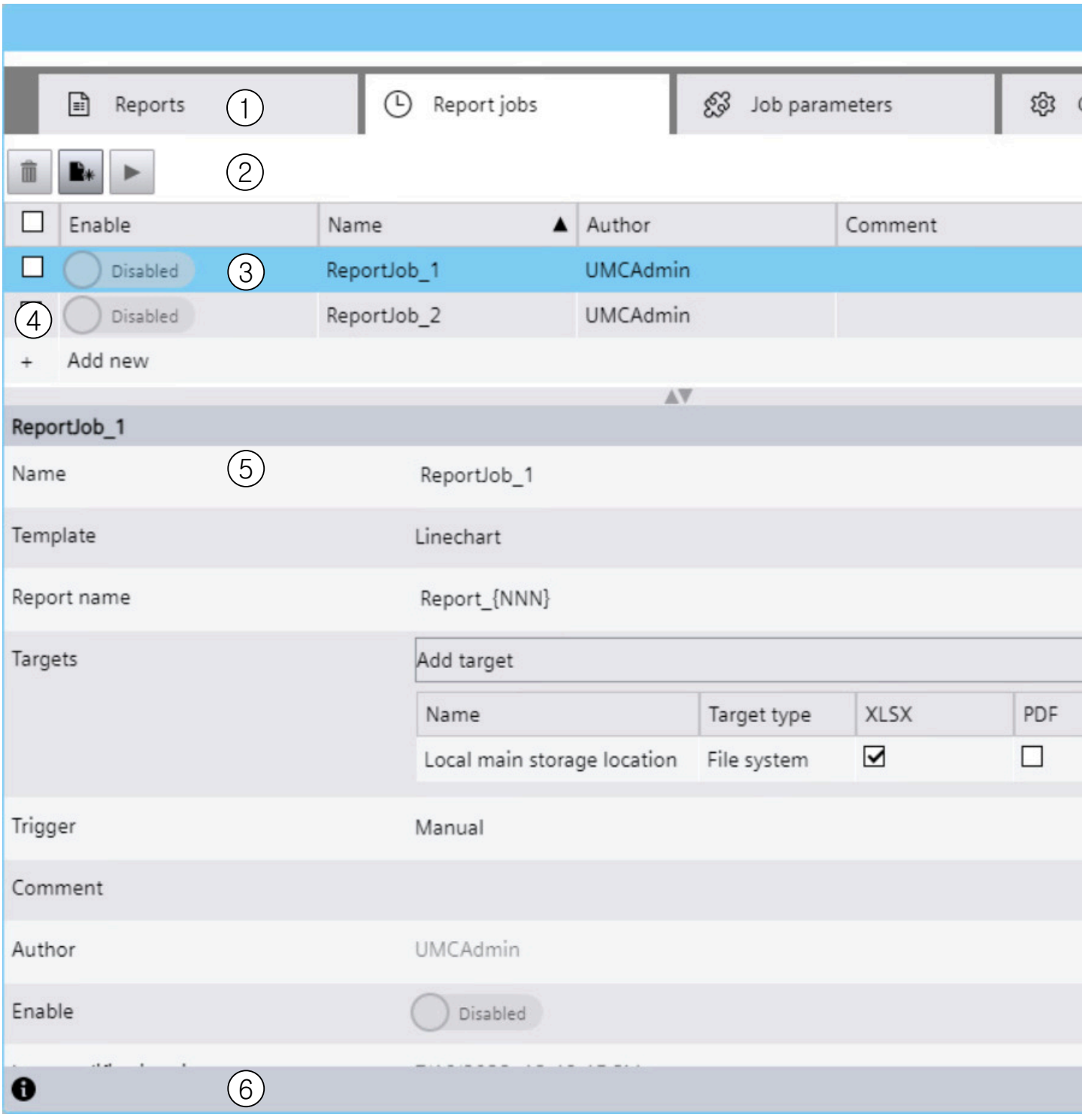
Note
Automatic data transfer

Changes in the "Reports" control are saved automatically.

Layout

You create and manage report jobs in the "Reports" control. You also have access to the reports generated by the report jobs.

The control has the following structure:



1 Tab for the configuration and management of reports, report jobs, job parameters and global settings

2 Toolbar
The buttons you see depend on the tab.

3 Work area
On the "Reports", "Report jobs" and "Job parameters" tabs: List of elements available on the tab
On the "Global settings" tab: The settings

4 Options for selecting the elements
You can select elements individually or all at once.

5 Detail area
Shows the properties of the selected element.

6 Information bar

Tab

"Reports" tab

Here you can see which reports have been generated. You can download or delete reports via the toolbar.

The "Status" column shows Information:

- On the status of the generated report files (XLSX and PDF)
- On the status of the targets (File system and E-mail)

Overview of the status icons:

Status	Description
	Execution has been successfully completed.
	An error occurred during execution.
	Execution is in progress.

A click on an icon opens a detailed status message.

"Report jobs" tab

Here you create new report jobs, manage existing report jobs or start a report job manually.

"Job parameters" tab

Here you manage the parameters with which you configure the report jobs in the "Report jobs" tab.

"Global settings" tab

Here you make the following settings:

- For sending e-mails
- For transfer of the control configuration
- For creating an offline configuration file
- For configuring paging

Toolbar

The following buttons are available in the toolbars of the tab:

Icon	Button	
	Delete	Deletes the elements whose option is enabled in the work area.
	• Add new • Import	• Creates a new element. • "Job parameters > Templates" tab: To import a report template into Runtime
	Run	In the "Report jobs" tab. Manually creates a report for the report job whose option is enabled in the work area.
	Export	• In the "Job parameters > Templates" tab: To export report templates • In the "Reports" tab: To download reports to the client

Information bar

The button in the information bar displays general information sent by the reporting service, for example, on whether a report job has been executed.

Setting global email settings

If configured accordingly, an e-mail is sent automatically after a report job is executed. The e-mail can include the report as an attachment.

Maintenance of the basic settings for sending e-mails is carried out in the "Global settings" tab:

- If necessary: The certificates that the e-mail sender uses to authenticate itself at the SMTP servers.
- The contact information of the e-mail senders and e-mail receivers.
- The configuration of the SMTP server via which the e-mails are sent.

Upload certificates

Store the certificates of the SMTP servers that require authentication via certificate.

Requirement

- You have access to the storage location of a valid certificate file.

Procedure

1. In the "Reports" control, click on the "Global settings > Certificates" tab.
2. Click "Add new" in the toolbar.
   Alternative: In the work area, click "Add new".
3. In the dialog that opens, select the certificate file.
4. Confirm your input.
5. Optional: Select the uploaded certificate in the work area and enter a comment on the certificate in the detail area.

Result

The certificates uploaded here are available in the "Contacts" tab.

Maintaining contacts

Store the data of the e-mail senders and email recipients.

Procedure

To create a new contact, follow these steps:

1. In the "Reports" control, click on the "Global settings > Contacts" tab.
2. Click "Add new".
3. Enter the name of the contact.
4. Enter the e-mail address of the contact.
5. To use the contact as a sender for an SMTP server that requires authentication with a certificate, select the appropriate certificate under "Certificate".
6. To use the contact as a sender for an SMTP server that requires authentication with a user name and password, enter the password.
   The e-mail address is used as the user name.
7. (Optional) Enter a comment relating to the contact.

Result

The contacts configured here are available:

- As the e-mail sender in the SMTP server configuration.
- As an e-mail recipient when configuring "target" job parameters with the target type e-mail

Maintenance of the SMTP server configuration

Store the data of the SMTP servers via which the e-mails are sent.

Requirement

Contacts that are suitable as senders have been entered in the "Global Settings > Contacts" tab.

Procedure

To create a new SMTP server configuration, follow these steps:

1. In the "Reports" control, click on the "Global settings > SMTP" tab.
2. Click "Add new".
3. Specify the following:

Field	Description
"Name"	Enter the name of the SMTP server configuration.
"Address"	Enter the URL of the SMTP server. Servers without authentication (e.g. company-internal mail servers) and with authentication are permitted. Example: URL of a company mail server: `mail.<Company name>.com`
"Port"	Enter the port number of the SMTP server. Default setting: 25
"Sender"	In the list, select the contact that is used as the sender for this SMTP server configuration. All contacts maintained under "Contacts" are offered to you for selection. Select a sender that meets the respective requirements of the server.
"Comment"	(Optional) Enter a comment relating to the SMTP server configuration.

Result

The servers configured here are available when configuring the "Target" job parameters with the target type email.

Configuring job parameters

Job parameters define the details of a report job.

You configure the following parameters on the "Job parameters" tab:

- Templates
- Trigger
  Define trigger when a report job is executed.
- Targets
  Targets define how a report is made available to users. The following target types are available:
  - "E-mail"
    An e-mail is sent after a report job is executed. The report generated by the report job can be included with the e-mail as an attachment.
  - "File system"
    The reports generated by the report job are stored in the file system.

The parameters configured here are available to you for selection when configuring the report jobs in the "Report jobs" tab.

You define the remaining job parameters while configuring a report job in the "Report jobs" tab.

See also

Importing and exporting report templates (Page 7368)

Deleting templates (Page 7369)

Configure trigger (Page 7369)

Add target with target type "E-mail" (Page 7372)

Importing and exporting report templates

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Templates" tab is visible in the control.
- Import: You have access to the storage location of the report template.
- Export: Report templates have been imported into the control.

Importing report template

1. Click "Add new" in the toolbar.
   Alternative: In the work area, click "Add new".
2. In the dialog that opens, select the file of a report template.

3. Confirm your input.

Note

No validation

The template is not validated during import.

4. Optional: In the work area, select the imported report template in the work area and enter a comment describing the template in the detail area.

Exporting report templates

1. In the work area, select the options next to the report templates you want to export.
2. Click "Export" in the toolbar.

The report templates are downloaded to the download folder or a user-defined directory according to the device settings.

Deleting templates

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Templates" tab is visible in the control.
- Templates have been imported into the control.

Procedure

1. In the work area, select the options next to the templates you want to delete.
2. Click "Delete" in the toolbar.

Deleting used templates

The "In use" column shows whether the template is used by a report job.

If you delete a template that is used by a report job, the report job is marked as inconsistent and no longer executed.

Configure trigger

Introduction

In the "Job Parameters > Triggers" tab you configure which automatic triggers are available for selection when configuring report jobs.

Report jobs with automatic triggers are executed if the report jobs on the "Report jobs" tab are set to active and their trigger event occurs. Users can also start the execution manually.

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Trigger" tab is visible in the control.
- To use the trigger type "Context trigger": Contexts are available in the project.

Add trigger

1. In the work area of the tab, click "Add new".
   A new trigger is created and displayed in the detail area.
2. Assign a unique name to the trigger.
3. Select the trigger mode:

Trigger type	Triggering the trigger
"Tag trigger"	Automatically when the configured value condition occurs at the tag defined in the trigger.
"Serial trigger"	Automatically within the user-defined interval when the time defined by the series has been reached.
"Context trigger"	Automatically when the selected context is started or stopped. Optional: By using a condition, you can also limit the triggering of the trigger to specific context values.

4. Depending on the selected trigger type, set the settings for the new trigger as described below.
5. Optional: Enter a comment for the trigger.

Settings for tag trigger

1. Click "Select tag".
2. Click "Load".
3. Select the required tag and click "OK".
4. Set the condition and the condition value.
   Example:

Set tag	<tag name>
Condition	>
Condition value	100

The trigger will be initiated when the tag receives a value greater than 100.

Settings for serial triggers

1. Select the serial pattern.
   The series pattern defines the occurrence and time at which the trigger is initiated.
   Example: Weekly > Every 2 weeks > Fridays
2. Select the series area.
   The series range defines the period in which the trigger is initiated.

"Start"	Specify the start date
"Time"	Specify the time at which the trigger is initiated.
"End on"	Specify the end date. The trigger will be executed for the last time on this day.
"End after"	Determine the number of dates after which the series ends.
"No end date"	The series runs indefinitely.

Settings for context triggers

1. Click "Select context".
2. In the "Context selection" dialog, click "Select plant object".
3. In the "Browser view" dialog, select a plant object and confirm your input.
   In the "Context selection" dialog you can see all contexts that have been defined for the selected plant object.
4. Select a context and confirm your input.
5. Under "Context status", select when the trigger will be triggered:

"Started"	When starting the context.
"Stopped"	When stopping the context.

6. Optional: To bind the execution of the report order to certain context values, you define a condition:

"Condition"	Select an operator.
"Value"	Select a context value.

Example:

Plant object	"MyPlant.hierarchy::PlantView/Bottling"
Context	"Product"
Context state	"Started"
Condition	=
Value	"Orange lemonade"

Report jobs with this trigger are always executed when the context "Product" defined on the plant object "Bottling" is started with the value "Orange lemonade".

Delete trigger

Select the option of the desired trigger in the work area of the "Job Parameters > Triggers" tab and click "Delete" in the toolbar.

Edit trigger

1. Enable the option of the desired trigger in the work area of the tab.
2. In the detail area, edit the settings of the trigger.

Note
No change of the trigger type

The trigger type can only be set when adding the trigger.

Add target with target type "E-mail"

Requirement

- The "Reports" control is placed on a screen of the project.
- The receivers of the e-mails are maintained as contacts in the "Global settings > Contacts" tab.
- An SMTP server, with which the e-mail is to be sent, has been configured in the "Global settings > SMTP" tab.

Procedure

1. In the "Reports" control, click on the "Job parameters > Targets" tab.
2. In the work area of the tab, click "Add new".
3. Select "E-mail" as target type.
   A new target is created and displayed in the detail area.
4. Assign a unique name to the target.
5. Select an SMTP server configuration.
6. Add the desired receivers and CC receivers:
   – To do so, select a contact from the list "Add receiver" or "Add CC receiver".
   – Add the contact by clicking "+".
7. Enter the e-mail subject.
   To integrate the report name into the subject line, use the placeholder {ReportName}.
8. Enter the e-mail text.
   To integrate the report name into the email text, use the placeholder {ReportName}.
9. (Optional) Enter a comment.

Result

The target is available for selection when configuring report jobs.

An e-mail is sent after a report job is executed with this target. The e-mail can include the report as attachment.

See also

Dynamic placeholder (Page 7386)

Add a target with "File system" target type

Introduction

A reporting job with a target of the "File system" target type saves reports to a file system.

When configuring the report jobs, you can choose from pre-configured and user-defined targets of this target type.

Preconfigured targets

The following targets with "File system" target type are pre-configured:

Local project storage location	The reports are stored in the following folder: `<Project folder of the Runtime project>\Reports`
Local main storage location	The reports are stored in the local main storage location for reports. The local main storage location is configured in TIA Portal in the Runtime settings of the HMI device. If this setting has not been set in TIA Portal, the reports are stored as follows: • Unified PC: In the folder configured during installation of Runtime or later in the "WinCC Unified Configuration" tool • Unified Comfort Panel: In the "Reports" folder on the SD card inserted in the Panel: `media/simatic/X51/Reports`

You can select these targets in the "Report jobs" tab. You cannot edit or delete these targets in the "Job parameters > Targets" tab.

User-defined targets

In the "Reports" control, you can create user-defined targets of the "File system" target type. These user-defined targets are always subfolders of the local main storage location.

Requirement

- The "Reports" control is placed on a screen of the project.
- Unified Comfort Panel: The panel contains the storage media configured in the TIA Portal Runtime settings as storage locations for reports and for the reporting database.

Procedure

To add user-defined targets of the "File system" target type, follow these steps:

1. In the "Reports" control, click on the "Job parameters > Targets" tab.
2. In the work area of the tab, click "Add new".

3. Select "File system" as target type.
   A new target is created and displayed in the detail area.
   Under "Destination path", you can see the path to the local main storage location for reports.
4. Assign a unique name to the target.
5. Under "Subfolder", enter the path to the subfolder in which the report is to be saved.
   Use the following notation: <folder name> or <folder name>\<folder name>\...

   Note
   Relative path information

   The path specification is relative to the local main storage location for reports.

6. (Optional) Enter a comment.

Result

The target is available for selection when configuring report jobs.

When a report job with this target is being executed, the generated report is stored in the subfolder of the local main storage location defined as the target. If the folder entered under "Target path" does not exist, it is created by the system.

Note
Change of the local main storage location for reports

When the local main storage location for reports changes, the targets are automatically adapted. New reports are stored relative to the new local main storage location. The old folders are not deleted.

Configuring report jobs

Creating a report job

Introduction

A *report job* is a job for generating reports in Runtime. The configuration of a report job controls the details of the generation.

Requirement

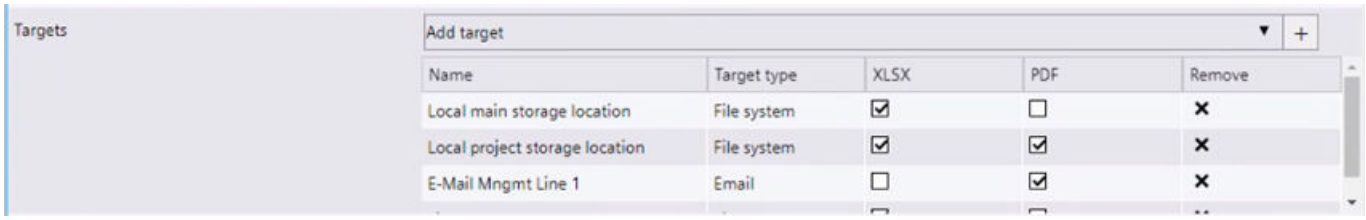
- The "Reports" control is configured on a screen of the project.
- The following job parameters were configured in the control:
  - At least one template has been imported.
  - To automatically execute a report job: Triggers are configured in the "Job parameters > Trigger" tab.

- For sending an email after execution of the report job:
  - Email contacts were configured in the global settings.
  - An SMTP server was configured in the global settings.
  - A target of the target type "E-mail" was configured in the "Job parameters > Targets" tab.
- For a report job with the target format PDF:
  - Microsoft Office Excel or LibreOffice is installed on the runtime server.
  - Depending on whether Excel or LibreOffice is installed, the information required for PDF creation was provided during the Runtime installation or in the "WinCC Unified Configuration" tool.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. Select "Add new" in the work area or click "Add new" in the toolbar.
3. In the detail area, enter a name for the report job.
4. Select a report template.
5. Configure the report name. See section Configuring report names (Page 7377).
   The configuration is applied to all reports generated by the report job.

6. Under "Targets", you determine how the reports are to be made available to users. Follow these steps:
   - Click "Add target".
     You see the targets configured in the tab "Job parameters > Targets".
   - Select a target.
   - Add the target by clicking "+".
     The target is added to the table to define the target formats.



   - Determine the formats in which the reports generated by the report job are provided for the target. In the table, activate the options of the desired formats for each target.

     Note

     Sending emails without a report

     If you deactivate both options for targets with "E-mail" target type, an email without attachment is sent after the report job has been executed.

     Note

     PDF as target type

     Generating PDFs with Excel is significantly slower than with LibreOffice. To generate large PDF reports, it is therefore recommended that you install LibreOffice.

     A PDF report created by LibreOffice can deviate in content or layout from a PDF report generated with Excel, for example, if the report template uses common Excel features that LibreOffice does not support, such as special fonts or chart types.

   - To remove a target from the report job, click the "Remove" button in the table.
7. Under "Trigger", select which event triggers the execution of the report job:
   - If the report job is only to be executed manually, select "Manual".
   - If the report job is to be executed automatically, select one of the other triggers configured under "Trigger".

     Note

     You can also execute the report job manually.

8. (Optional) Enter a comment for the report job.
9. Specify whether the automatic execution of the report job is active or paused. To do this, set the slider "Enabled" or "Disabled".

   Note

   You can still execute disabled report jobs manually.

Result

The report job is saved automatically.

When its trigger occurs, the report job is executed. A report is generated and made available as configured under "Targets".

See also

Execution of report jobs (Page 7381)

Configure trigger (Page 7369)

Add a target with "File system" target type (Page 7373)

Add target with target type "E-mail" (Page 7372)

Tips on design and layout (Page 7449)

Managing report jobs

Requirement

- The "Reports" control is configured on a screen of the project.
- Report jobs have been configured in the control.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. To edit a report job, proceed as follows:
   - Select the report job in the work area.
   - In the detail area, edit the settings of the report job.
     You have the same options as when creating a report job.
3. To delete report jobs, proceed as follows:
   - In the work area, enable the options next to the report job.
   - Click "Delete" in the toolbar.

Configuring report names

Note

Make sure that the generated report name does not violate the policy of the operating system regarding the maximum length of file names.

Introduction

The default name of reports is `Report_{NNN}`.

To use different report names, enter one or more placeholders at the report job. The placeholders are combined to form the report name during execution of the report.

Placeholder types

Placeholders have one of the following types:

Placeholder type	Description	
Text	For user-defined fixed texts	
Counter	On automatic numbering	Dynamic placeholders The placeholders are broken down into values during execution of the report.
Date format	For outputting the generation time	
Tag	To output the process value of an online tag	

Unique report names

If the report name uses counter or date format placeholders, the report job generates unique report names.

Requirement

- The "Reports" control contains a screen of the runtime project that is running.

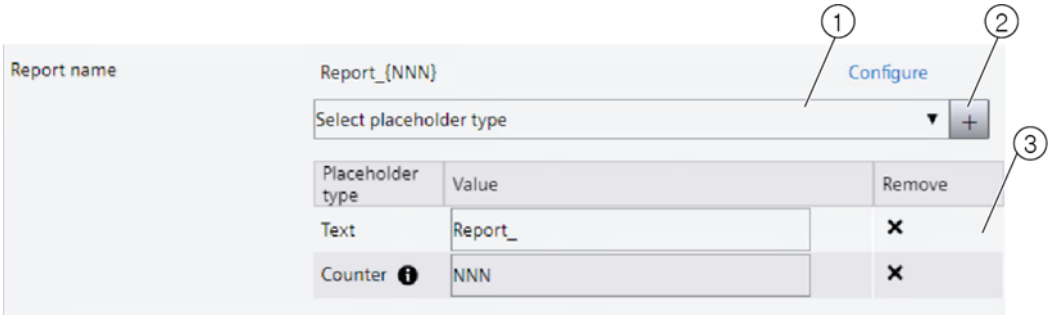
Procedure

You can enter the placeholders manually in the "Report name" field or you can have the software help you configure the report name.

To have the software help configure the report name, follow these steps:

1. Select the "Report jobs" tab in the "Reports" control.
2. Select a report job in the work area.
   You can see the settings for the report job in the detail area.

3. Next to "Report name", click "Configure".
   You see the following operator controls:



① List for selection of the placeholder type
② Button for adding a placeholder of the selected type
③ Table for configuring or removing the placeholder

Note

For the default report name, the "Report name" has the value `Report_{NNN}` and the table shows the placeholders "`Report_`" and "`NNN`".

To swap the order of placeholders or to add a placeholder in the center, delete the placeholders and then add them in the desired order.

4. Optional: To delete the default placeholders, click "x" in the placeholder table.
5. Select the desired type under "Select placeholder type".

Note

A report name can contain only one counter.

6. Click "+".
   An empty placeholder of the corresponding type is added at the end of the table.
7. Enter the placeholder under "Value" in the placeholder table.

Placeholder type	Description	Example
"Text"	Enter the text.	`Report_`
"Date format"	Enter a date placeholder.	A list of permitted placeholders and examples can be found in section Dynamic placeholder (Page 7386).
"Counter"	Enter a counter placeholder.	
"Tag"	Enter the full name of an online tag.	`RT1_Brewery::BatchNo`

Note

Enter the dynamic placeholders without any markup characters.

Alternatively, you can select an online tag via the user interface. Follow these steps:

– Click the "..." button on the tag placeholder.

– In the "Tag selection" dialog, click the "Search" button.
  You can see all the tags of the Runtime project that is running.

  Note

  Scrolling and filtering

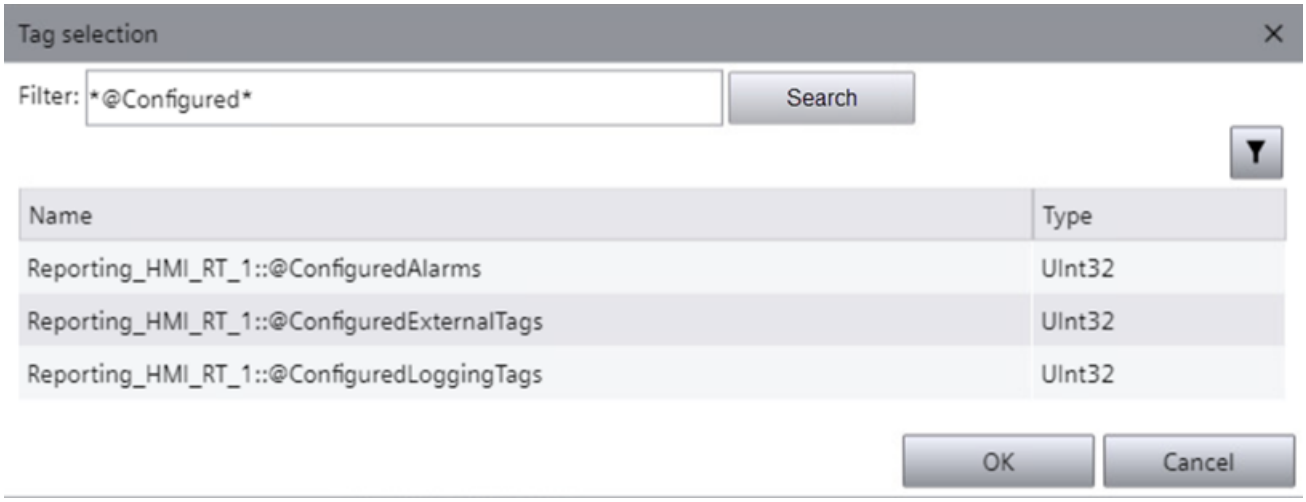
  Use the page navigation buttons to scroll forward or backward.

  To filter the displayed tags, enter a filter string in "Filter" and click "Search".

  You use the wildcard "*" to filter by partial strings.

  For example:
  - *T* returns all tags with a "T" in their name.
  - *T returns all tags that end in "T".
  - T* returns all tags that start with "T".

  When filtering for structures, the separators must be part of the filter string.



– Click the desired tag.

– Confirm with "OK".

In the "Report name" field, the placeholder you added is appended to the end of the report name.

Alternative procedure

To enter the placeholders manually, proceed as follows:

1. Select the "Report jobs" tab in the "Reports" control.
2. Select a report job in the work area.
   You can see the settings for the report job in the detail area.
3. Enter the desired combination of fixed texts and dynamic placeholders manually in the "Report name" field.
   Use markup characters for the dynamic placeholders. See section Dynamic placeholder (Page 7386).

Example:

"Report name" value	Generated report name
`Report_{yyyymmdd}_{HHMMss}_{@PC1_Brewery::BatchNo}`	`Report_20190101_170001_BatchNo_87002314`

Result

When generating a report, the dynamic placeholders are resolved and all placeholders are merged to form the report name.

If a process value contains a character that is not permitted in file names, it is replaced by an underscore.

If there is an error resolving the name, e.g. because the tag is not found in runtime, the tag placeholder in the name is replaced by `ERR`. The process is logged in the generation status of the report.

Execution of report jobs

Automatic and manual execution

Automatic execution

Report jobs that have a tag trigger, serial trigger or context trigger and are set to active on the "Report jobs" tab are automatically executed when their trigger occurs.

Manual execution

Report jobs with a trigger of the "Manual" type must always be executed manually.

In addition, you can at any time manually execute report jobs that have a tag trigger, serial trigger or context trigger.

System response to errors

- Error adding the report job to the queue
  The execution of the report job is discarded. A system alarm documents the error.
- Error executing the job
  In the "Reports" control, "Reports" tab, the status icon indicates the error. A click on the icon opens a detailed status message.
  A system alarm documents the error.

See also

Running a report job manually (Page 7382)

Configure trigger (Page 7369)

Running a report job manually

You can execute report jobs manually at any time, regardless of their trigger type. This also applies to report jobs that were disabled in the "Report Jobs" tab and whose automatic execution is therefore paused.

Requirement

Report jobs have been configured in the "Reports" control.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. In the work area, enable the option next to the report job that you want to execute manually.
3. Click "Execute" in the toolbar.

Result

The report is generated. You can download it in the "Reports" tab.

Downloading reports

You can download the reports stored by the report job in the file system to your device.

Depending on which file formats have been set in the report job, you can download the report as an XLSX file and as a PDF file.

Requirement

- Report jobs with the target type "File system" have been configured and executed in the "Reports" control.

Procedure

1. Select the "Reports" tab in the "Reports" control.
2. In the work area, select the option in the left column for each report that you want to download.
3. Enable the desired target formats in the "Files" column.

   Note

   Generation status

   You are only offered successfully generated formats.

   In the "Status" column you can check whether the generation for a format has failed. For a detailed status message click on the icon of a target format.

4. Click "Export" in the toolbar.

Result

The reports are downloaded into the download directory of the browser.

You can edit, distribute, or log the reports.

See also

Installation of the Reporting add-in (Page 7391)

Exporting an offline configuration file

An offline configuration file is required to configure reporting templates in the Reporting Excel add-in without an online connection to the Runtime server.

Requirement

- The "Reports" control is placed on a screen of the project.
- The Runtime project has data that can serve as data source elements in the reporting template, such as alarms and logging tags.

Procedure

1. In the "Reports" control, click on the "Global settings > Configuration" tab.
2. Enter the name of the offline configuration file under "Offline-configuration".
3. Click "Export offline configuration".

Result

A JSON file with the data source elements of the Runtime project is created. The file is downloaded to the download folder or a user-defined directory according to the device settings.

You can select the configuration file in the Reporting Excel add-in as data source for an offline connection.

Transferring the control configuration

You have the option of reusing the settings in the "Reports" control, for example, on a device in another network. To do this, export the existing configuration on the one device from the control to a ZIP file. Then import the file into a "Reports" control on the other device.

Scope

The transfer covers the following data:

- Global settings, without passwords and certificates
- Job parameters, including the report templates available in the control
- Report jobs

Reports are not transferred.

Requirement

- The "Reports" control is placed on a screen in the project running in Runtime.
- Export: Settings have been made, e.g. contacts maintained, report templates imported, and report jobs created in the "Reports" control.
- Import: You have access to the ZIP file generated by the export on the device on which Runtime is installed.

Export configuration

1. In the "Reports" control, select the "Global settings > Configuration" tab.
2. Enter the name of the export file under "Export/import configuration > Export".
3. Click "Export configuration".

The configuration is exported to a ZIP file and downloaded to the default download directory of the device.

Import configuration

1. In the "Reports" control, select the "Global settings > Configuration" tab.
2. Click "Select import file" under "Export/import configuration".
3. Select the ZIP file in File Explorer and confirm your selection.
4. Runtime checks whether the control already contains configurations:
   - No: The configuration is imported.
   - Yes:
     Select "OK" to import the configuration. The existing configuration is overwritten.
     Select "Cancel" to cancel the import.

Configuring enable paging

To set how many entries the lists in the work area of the "Reports" control display per page, follow these steps:

1. In the "Reports" control, click on the "Global settings > Configuration" tab.
2. Under "List Settings", select the number of entries.

If a list has more entries, these are split over several pages. Use the buttons below the list to switch pages.

Note

The setting is lost through a screen change.

Inconsistencies and error diagnostics

Note

Inconsistent report jobs are not executed.

The templates available in the "Reports" control are not validated.

Display of inconsistencies and errors

Errors and inconsistencies are displayed as follows:

In the control	If job parameters are affected. Examples: • No template is set for a report job. • A tag that triggers a report job is deleted in the engineering system. The project is reloaded into the device.
In the "Error log" worksheet of the report	Errors or inconsistencies affecting the content of the report. Example: The report evaluates data from a tag that is no longer available in runtime.
As system alarm	For errors and inconsistencies that do not affect job parameters or the contents of the report. Example: The ExecuteReport system function transfers a report job that does not exist.

Job parameters

The following values lead to errors and inconsistencies:

Parameter	Invalid values	Default setting
"Name"	Zero, empty or already assigned name	"New report job"
"Template"	Zero, empty or "None". Name of a template that is not imported	"None"
"Target name"	Zero or empty	"NewReportJob[NN]"

Dynamic placeholder

Introduction

Dynamic placeholders are evaluated when the report job is executed and replaced with text in runtime.

The following job parameters can contain placeholders:

- Report name
- Targets with the target type "E-mail": Subject and text of the email

Dynamic placeholders for report names

Use dynamic placeholders for counters and/or dates to generate unique report names:

Counter placeholder	Description	Example		Area
		Configuration	Result	
{N}	Automatic numbering	Rep_{N}	Rep_1	0...9
{NN}		Rep_{NN}	Rep_01	00...99
{NNN}		Rep_{NNN}	Rep_001	000...999

Date placeholder	Description	Example		Area
		Configuration	Result	
{yy}	Current year	Rep_{yy}	Rep_18	Valid year with 2 digits
{yyyy}		Rep_{yyyy}	Rep_2018	Valid year with 4 digits
{m}	Current month	Rep_{m}	Rep_1	Valid month, no prefixed 0 for months in single-digit range
{mm}		Rep_{mm}	Rep_01	Valid month, prefixed 0 for months in single-digit range
{mmm}		Rep_{mm}	Rep_Jan	Month abbreviation with 3 characters
{mmmm}		Rep_{mmmm}	Rep_January	Month with full name

Date placeholder	Description	Example		Area
		Configuration	Result	
{d}	Current day of the month	Rep_{d}	Rep_1	Valid day, no prefixed 0 for days in single-digit range
{dd}		Rep_{dd}	Rep_01	Valid day, prefixed 0 for days in single-digit range
{ddd}		Rep_{ddd}	Rep_Mon	Day abbreviation with 3 characters
{dddd}		Rep_{dddd}	Rep_Monday	Day with full name
{h}	Current hour	Rep_{h}	Rep_1	Current hour (12-hour clock), no prefixed 0 for single-digit values
{hh}		Rep_{hh}	Rep_01	Current hour (12-hour clock), prefixed by 0 for single-digit values
{H}		Rep_{H}	Rep_13	Current hour (24-hour clock), no prefixed 0 for single-digit values
{HH}		Rep_{HH}	Rep_13	Current hour (24-hour clock), prefixed by 0 for single-digit values
{M}	Current minute	Rep_{M}	Rep_6	Valid minute, no prefixed 0 for single-digit values
{MM}		Rep_{MM}	Rep_06	Valid minute, prefixed by 0 for single-digit values
{s}	Current second	Rep_{s}	Rep_41	Valid second, no prefixed 0 for single-digit values
{ss}		Rep_{ss}	Rep_41	Valid second, prefixed by 0 for single-digit values

Use a dynamic placeholder for tags to integrate process values in the report name:

Tag placeholder	Description	Example		Area
		Configuration	Result	
{@<Full Tag name>}	Process value of an online tag	Rep_{@PC1_LineA::MyTag1}	Rep_On	Process value of the online tags If the value contains a character that is not permitted in file names, it is replaced by an underscore. If there is an error resolving the name, e.g. because the tag is not found in runtime, the tag placeholder in the name is replaced by `ERR`. The process is logged in the generation status of the report.

Examples:

Definition with placeholder	Generated report name
LineA_{yyyymmdd}_{HHMMss}	LineA_20190101_170001
LineA_{yymmmd}_{hhMMss}	LineA_19Jan1_050001
LineA_{NNN}	LineA_014
LineA_{yyyymmdd}_{HHMMss}_BatchNo_{@PC1_Brewery::BatchNo}	LineA_20190101_170001_BatchNo_87002314

Placeholder for email subject and email text

To integrate the report name into the subject line or the email text, use the following dynamic placeholder {ReportName}.

Markup

Use the following markup characters for dynamic placeholders:

- Placeholders for counter and date: `{ }`
- Placeholders for tags: `{@}`

Note

There is no markup in the placeholder table for defining the report name. See also section Configuring report names (Page 7377).

Reporting system events

Reporting system events

The most important system events are listed below.

ID	Alarm text	Effect/causes	Solution
538640385	Initialization of the reporting service failed	Initialization of the reporting service fails.	Contact Siemens customer service.
538640386	Report Data Provider cannot be started	The data provider for reports could not be started.	Contact Siemens customer service.
538640387	The report cannot be started for the job [name].	The Report Creator for report jobs cannot be started.	Check the report job settings. If you use the "ExecuteReport" system function, check the name of the report job and the parameters passed when calling the function.
538640388	An error occurred during communication with the database server	The reporting database cannot be found or access is not possible for other reasons.	Check whether the reporting database is available at the storage location configured in the Runtime settings in engineering. Example for panel: • Is the SD card plugged in? • Does the folder specified as storage location in the Runtime settings exist? • Has the folder been specified in the correct notation?
538640389	The creation of the report job [name] failed	The Report Creator is missing information about the report job. A possible reason for this are problems with processing the report template.	Check the report job settings and the report template.
538640390	Report failed	Report Creator reports an error while generating the report.	Check the detailed error message for the report: Control "Reports" > "Reports" tab > "Status" column.

Configuring report templates in the add-in

Requirements

General requirements and restrictions

Installing the Excel add-in

The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

Update the operating system and the Excel version if necessary.

To install the add-in with a local Excel installation, MS Excel with build 16.0.6769 or higher is required.

Note

Note the Microsoft upgrade restrictions

If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

IIS settings for standalone installation of the Excel Add-In

To install the Excel Add-In on a PC without Unified Runtime, the same IIS (Internet Information Services) settings must be active in Windows that are required to install WinCC Unified Runtime on a PC.

You can find additional information in the "SIMATIC Unified PC Installation" user help section on the software and hardware requirements.

Unified Comfort Panel

The following restrictions apply to generating reports on Unified Comfort Panels:

Contexts	Contexts are not supported in V17 for Unified Comfort Panel. This option is not available in a report template with a Unified Comfort Panel as data source. When you generate a report on a Unified Comfort Panel whose report template uses this option, error entries are generated in the "ErrorLog" worksheet of the report.
Storage location of the Reporting database	The following folder on the SD card plugged into the panel is preconfigured as the storage location of the reporting database: • Device version V18: `media/simatic/X51/Reports` You can configure and load another storage location in TIA Portal in the Runtime settings of the Panel. • Device version <V18: `media/simatic/X51`
Storage location for reports	The "Reports" folder on the SD card inserted in the Panel is permanently pre-configured as the local main storage location for reports: `media/simatic/X51/Reports` For Panels with device version V18, you can configure and load a different local main storage location in the Runtime settings of the Panel in TIA Portal.

Enable Reporting

The use of Reporting requires that the Reporting functionality was enabled:

- For configuring report templates:
  Reporting must be enabled for the Runtime project that serves as the data source.
- For configuring report jobs and generating reports in Runtime as well as in a simulation:
  Reporting must be enabled for the Runtime project that is running on the HMI device or is being simulated.

The reporting functionality of a Runtime project is enabled in TIA Portal in the Runtime settings of its HMI device with the option "Enable reporting".

Note

Devices with a device version lower than V18

Reporting is always enabled for HMI devices with a device version lower than V18.

See also

Version compatibility (Page 7359)

Installation of the Reporting add-in

Note

Regular updates of operating system and MS Excel

The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

Update the operating system and the Excel version if necessary.

To install the add-in with a local Excel installation, MS Excel with build 16.0.6769 or higher is required.

Note

Note the Microsoft upgrade restrictions

If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

Procedure

1. Install the Excel manifest on the computer.
2. Set up read access to the installation path of the Excel manifest.
3. Add the add-in to Excel.

See also

Installing the Excel manifest (Page 7392)

Setting up read access to the Excel manifest (Page 7392)

Adding the Reporting add-in in Excel (Page 7393)

Installing the Excel manifest

Procedure

1. In the installation package of WinCC Unified on "DVD_2", double-click the file "Support\Reporting\SIMATIC_WinCC_Unified_Reporting_<Version number>.exe".
2. Select the target directory to which the underlying ZIP file is extracted and confirm your input. The ZIP file is extracted and setup starts automatically.

   Note

   Start setup manually

   To start the setup manually after the file was extracted, select the option "Extract the setup files without being installed".

   Start the setup later by running the "Setup.exe" file as administrator in the target directory.
3. Follow the setup instructions.
4. In the "Configuration" step, select the option for the Excel add-in.
5. Click "Next" and follow the setup instructions.

See also

Installation of the Reporting add-in (Page 7391)

Setting up read access to the Excel manifest

Requirement

The Excel manifest is installed.

Procedure

Give the users that create templates with the Excel add-in read access to the installation path of the Excel manifest: <target directory>\WinCCUnifiedReporting\Excelmanifest

Note

This step is also necessary if the user belongs to a group in the user management with general read permission.

See also

Installing the Excel manifest (Page 7392)

Installation of the Reporting add-in (Page 7391)

Adding the Reporting add-in in Excel

Requirement

- The Excel manifest is installed on the PC.
- Read access to the installation path of the Excel manifest is set up.
- The following software is available on the computer:
  - Local Excel
    MS Excel (Build 16.0.6769 or higher)

    Note

    Regular updates of operating system and MS Excel

    The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

    If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

    Update the operating system and the Excel version if necessary.

    Note

    Note the Microsoft upgrade restrictions

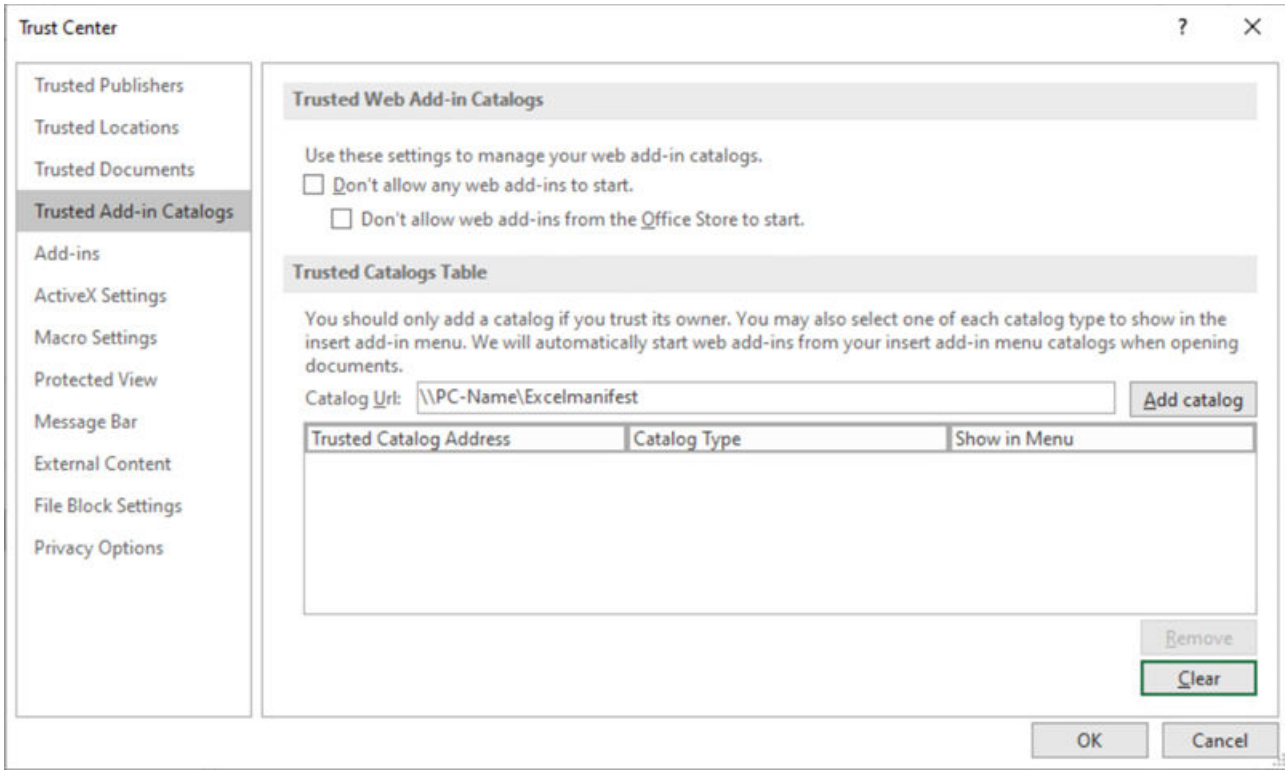
    If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

  - Or Office online

Procedure

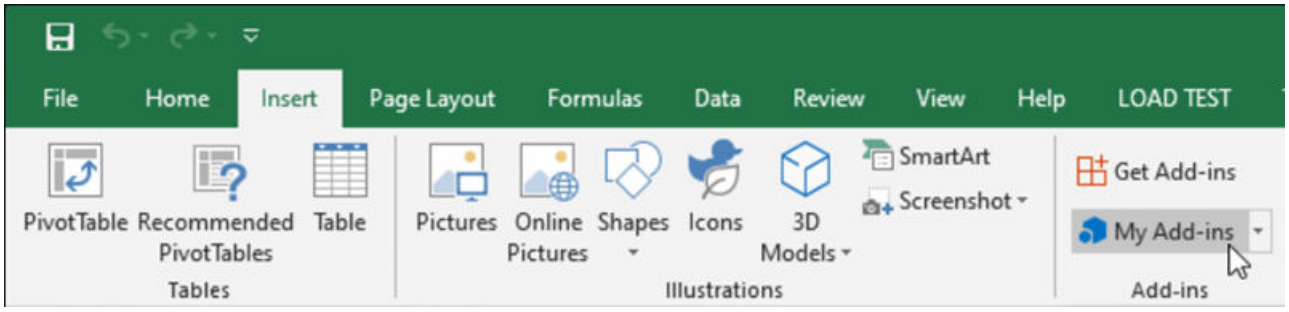
1. Open Microsoft Excel.
2. Open the "Trust Center" under "File" > "Options".
3. Click "Trust Center Settings".
4. Click "Catalogs of trusted add-ins".

5. Add the catalog using the URL "\\<Computer name>\Excelmanifest".



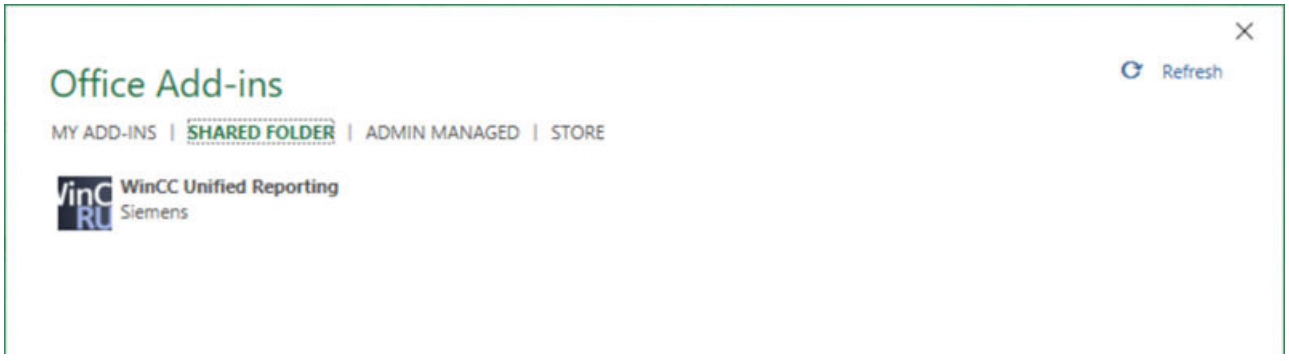
6. Make sure that the check mark in the "Show in Menu" column is set.
7. End and restart Excel.

8. In the "Insert" menu, click "My Add-ins".



In the "Office Add-ins" dialog box, the Siemens add-in is displayed under "Shared folders".

9. Select the add-in and click "Add".



See also

Installing the Excel manifest (Page 7392)

Setting up read access to the Excel manifest (Page 7392)

Installation of the Reporting add-in (Page 7391)

Configuring Internet Explorer and Edge

The Reporting Excel add-in uses the certificate that was selected during installation of WinCC Unified Runtime or later in "WinCC Unified Configuration".

Some browsers do not recognize self-signed certificates as trusted. If you use a self-signed certificate for WinCC Unified Runtime, you must add the certificate to the list of trusted certificates in Internet Explorer or Edge on the device on which the Excel add-in is installed.

You can find detailed information on handling certificates here.

Trusting self-signed certificates

The following section describes the procedure for adding a self-signed certificate to the list of trusted certificates, using Internet Explorer as an example:

1. Start Internet Explorer.
2. In the address line, enter the host name entered when creating the certificate.
   You will receive a security warning.
3. Click "Continue to this website (not recommended)".
4. Click "View Certificates".
5. Click "Install Certificate".
6. Click "Place all certificates in the following store" and "Browse".
7. Click "Trusted Root Certification Authorities" followed by "OK".

   Note

   Do not use the preset options for automatic selection of the certificate store.

8. Exit the dialog.
9. If you receive a security warning as to whether you want to trust the certificate, confirm it with "Yes".
10. Load the page again.

Login

A login dialog opens in the Excel add-in in the following cases:

- After start of Excel and the add-in
- When using an online connection: When the connection to the Runtime server must be re-established.
  Examples:
  - Runtime has been reloaded.
  - The security token has expired due to a timeout.

Requirement

- The add-in is installed.
- When using an online connection:
  - A Runtime server is accessible.
  - A Runtime project is running on the server.

Procedure

In order to use an online connection, log onto a Runtime server:

1. Under "Server", enter the name of the server on which the project that is to serve as data source for the report template is running.
   Use the same spelling as when the Runtime server certificate was created.

   Note

   If Runtime is installed on the same computer as the add-in, use of the name "localhost" is not permitted.

2. Enter the user name and password of a user that is registered on the server in the Runtime user management.
3. Click "Login".

In order to use an offline connection, click "Go offline".

Result

Online connection

The add-in is connected to the Runtime server and the options available there are loaded.

You can now create report templates.

Offline connection

Before you create report templates, set up the offline connection.

See also

Installation of the Reporting add-in (Page 7391)

Setting up an offline connection (Page 7400)

Setting up a data source

Using an online connection

When an online connection is present, the add-in establishes a connection to a Runtime server. The project running on the server serves as data source for the add-in.

The connection settings allow you to:

- Change the connected Runtime server to another Runtime server
- When a report template that was created with a different Runtime server than the currently connected server is reused: check the options available on the server and delete the options that were not loaded

Setting up an online connection

Requirements

- A Runtime server is accessible.
- A Runtime project is running on the server.

Procedure

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Click "Online" under "Connections" in the add-in.
3. Under "Server", enter the server name.
   Use the same spelling as when the Runtime server certificate was created.

   Note

   If Runtime is installed on the same computer as the add-in, use of the name "localhost" is not permitted.

4. Click "Load".

Result

- A server node is created.
- The add-in is connected to the Runtime server and its options are loaded.
  Data source items of these options can be added to report templates. Their data can be read in from Runtime to Excel.

  Note

  To check which options were loaded, click on the server node.

  Options that are being used in the currently open report template but are not available on the connected server have a red icon. You can remove the option:

- If no connection can be established or an incorrect server name has been entered, the add-in will display a corresponding error message.

See also

Removing options (Page 7398)

Removing options

Introduction

If you reuse report templates across servers, e.g. in order to adapt an existing template for another project, it may be necessary to remove unavailable options from the connection settings.

The procedure for this is presented using the Performance Insight option as an example.

Requirement

- The add-in was connected to a server on which the Performance Insight (PI) option is installed.
- A report template that uses KPIs was created with the add-in.
- The add-in was then connected to a server without the Performance Insight option installed for the purpose of adapting the template to the project running there.

Removing an option

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Under "Connections", click on "Online".
3. Select the server node.
   You see the loaded options under the server node:

	Available options The following applies to data source items of these options: • They can be added to the report template. • Their data can be read in from Runtime to Excel in the add-in.
	Unavailable options In the example: Performance Insight The following applies to data source items of these options: • They cannot be added to the report template. • If the report template already has a data source element of this option, its data cannot be read in from Runtime to Excel.

4. Select the "Performance Insight" option under the server node.
5. Click the "Delete" button next to the option.
6. Confirm your input.

Result

The option is removed from the connection settings.

Next, remove all data source items of this option from the report template.

Reloading an option

When the add-in is connected to a Runtime server, all options available on the server are loaded.

To reload an option that was deleted in the connection settings but is available on the server, select the server node and click "Load".

Using an offline connection

With the offline connection, the add-in uses a configuration file as data source.

The connection settings allow you to:

- Change the configuration file used
- When reusing a report template with a configuration based on a Runtime server different to that of the currently selected configuration file: Check the available options and delete the options that were not loaded.

Setting up an offline connection

Requirement

An offline configuration file was created in the "Reports" control in Runtime. The configuration file is available on the device.

Procedure

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Under "Connections", click on "Offline".
3. Click "Open offline configuration".
4. Select the desired file in the window that opens and confirm your entries.
5. Click "Load".
6. Select the desired options.
7. Confirm your entries.

Result

- A server node is created. The node bears the name of the server on which the configuration file is based.
- The configuration file, together with its options, is loaded into the add-in. The data of the configuration file is available for configuring the report template.

Note

To check which options were loaded, click on the server node.

Options that are being used in the currently open report template but are not available in the configuration file have a red icon. You can remove the option:

See also

Removing options (Page 7401)

Exporting an offline configuration file (Page 7383)

Removing options

Introduction

If you reuse report templates across servers, e.g. in order to adapt an existing template for another project, it may be necessary to remove unavailable options from the connection settings.

The procedure for this is presented using the Performance Insight option as an example.

Requirement

- The add-in was changed over to an offline connection whose configuration file does not include Performance Insight.
- A report template was opened in the add-in whose configuration is based on a connection to a Runtime server on which Performance Insight is installed.

Removing an option

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Under "Connections", click on "Offline".
3. Select the server node.
   You see the loaded options under the server node:

	Available options The following applies to data source items of these options: • They can be added to report templates. • Their data can be read in from the configuration file to Excel.
	Unavailable options In the example: Performance Insight The following applies to data source items of these options: • They cannot be added to the report template. • If the report template already has a data source element of this option, its data cannot be read in from the configuration file to Excel.

4. Select the "Performance Insight" option under the server node.
5. Click the "Delete" button next to the option.
6. Confirm your input.

Result

The option is removed from the connection settings.

Next, remove all data source items of this option from the report template.

Reloading an option

When a configuration file is loaded, all options available in the file are loaded.

To reload an option that was deleted in the connection settings but is available in the configuration file, select the server node and click "Load".

Configuring report templates

Requirement

An online connection or offline connection has been established.

Procedure

To create a new report template, proceed as follows:

1. Open a new Excel file.
2. Add a segment.
   You can choose between time series segments and single value segments.
3. Add data source items to the segment.
   The exact procedure depends on the type of the data source item.
4. Optional: If you do not want a data source item to use the default configuration, determine its configuration.
   You have the following options:
   - Select an existing configuration.
   - Create a new configuration and select it.
   - Define a local configuration.
5. Optional: To define additional segments, repeat steps 2 to 4.
6. Optional: When using an online connection, test the template by reading the runtime data of selected segments or all segments.

See also

Setting up a data source (Page 7397)

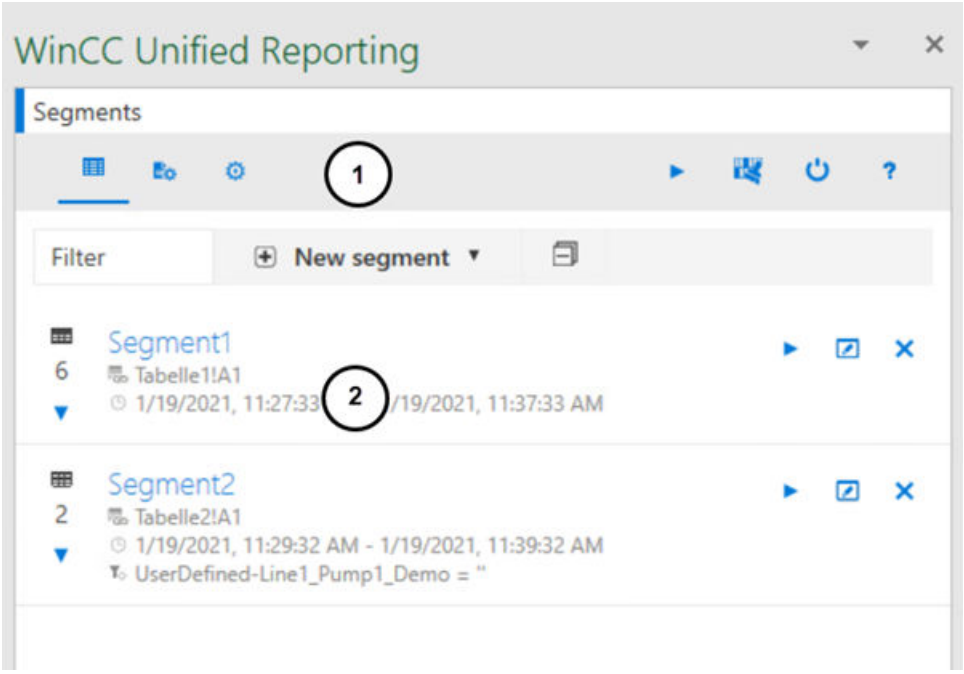
User interface of the add-in

Requirement

- The "WinCC Unified" tab is visible in Excel.

Structure

If you click on "Segments" in the "Configuration" group, you see the following interface:



① Toolbar
② Work area

Toolbar buttons:

Button	Tooltip	Description
	"Segment configuration"	Loads the interface to add and edit segments in the work area.
	"Data source item configuration"	Loads the interface for adding and editing the configuration of a data source item in the work area.
	"Basic settings"	Loads the interface for setting the language settings in the work area.
	"Update all"	Reads the Runtime data of the connected data source into the data tables of the segments.
	"Delete Runtime data"	Removes all Runtime data from the report template.
	Logoff	Logs out the user currently logged in to the add-in.
	Help	Opens the user help for the add-in.

See also

The segment user interface (Page 7408)

Working with segments

Basic information on segments

Definition

A report template consists of any number of segments. Each *segment* is a container to which you can add any number of data source items. The segment reads the data from its data source items.

There are time series segments and single value segments.

Note

Hierarchical segments of PI options

Hierarchical segments are also available with PI Options installed. For more information on this, refer to the PI Options help.

Time series segments

A *time series segment* documents the data of its data source items in a defined time period.

It has a legend table and a data table.

Data source items

Time series segments can have the following data source items:

- Logging alarms
- Alarm statistics
- Logging tags
- User-defined columns
- Contexts
- Audit

Note

Data source items of the PI options

If PI options are installed, additional data source items can be added. For more information on this, refer to the PI Options help.

Legend table

The table header row provides general information about the segment and its data source items.

You decide which type of information is provided when you create or edit the segment.

Data table

The data table outputs the data of the data source items. It documents how the data source items have changed in the defined time period.

The data table of a time series segment has the following columns:

Columns		Description
Time stamp column		Always output Always output as the first column
Per data source item	Standard column	The standard column provides the standard property of the data source item. This property depends on the type of data source item. For a data source item of the Tag type, e.g. the tag value
	Optional columns	Provide more information about the data source item. What information this is depends on the type of the data source item. For a data source item of the Tag type, e.g. the quality code of the tag value You change the default settings for visibility, column title and order of these columns in the configuration of the data source item.

In the default setting, the data source items in the data table have the order in which they were added to the segment.

Note

When the standard columns and optional columns provide numerical values, you can have the actual values replaced with texts or graphics from a text list or graphic list when importing the Runtime data.

Single value segments

A *single value segment* documents exactly one value for its data source items.

Data source items

Single value segments can have the following data source items:

- Logging tags
- Tags

Note

Data source items of the PI options

If PI options are installed, additional data source items can be added. For more information on this, refer to the PI Options help.

Data table

The data table of a single value segment has the following columns per data source item:

Columns	Description
Standard column	The standard column provides the standard property of the data source item. For tags and logging tags: the tag value
Optional columns	Provide more information about the data source item. For tags and logging tags: • Time stamp • Data source item • Quality code of the tag value You change the default settings for the visibility of these columns in the configuration of the data source item.

The data table of a single value segment shows the data source items in the order in which they were added to the segment.

Note

When the standard columns and optional columns provide numerical values, you can have the actual values replaced with texts or graphics from a text list or graphic list when importing the Runtime data.

Single row segments do not have a table header row. However, in the configurations of their data source items, you can determine whether a caption is inserted for the displayed columns and the position at which this occurs.

See also

Standard column (Page 7406)

Standard column

Introduction

For each data source item of a segment, a standard column is added in the data table of the segment.

Content of the standard column

The standard column provides the standard property of the data source item and depends on the type of the data source item:

Data source item type	Default column title	Value
Logging alarm	"Alarm ID"	Alarm IDs of the displayed alarms
Alarm statistics	"Alarm statistics [ID]"	Alarm IDs of the alarms displayed in the alarm statistics

Data source item type	Default column title	Value
Tag or logging tag	"<Name of the tags or logging tags>"	Value of the tag or logging tag
Context	"<Name of the context object>"	Context name
Audit	"Audit [<object name>]"	The name of the object monitored by the Audit Trail
User-defined column	Name entered when creating the data source item	As set in the configuration of the data source item: • A fixed string. Or • A dynamically calculated string

Changing the column title

You can replace the default column title with a localizable display name. See Setting a display name for standard column (Page 7443).

Replacing numerical values

If the standard column provides numeric values, you have the option to have the actual values replaced with texts or graphics from a text list or graphic list when the Runtime data is read in. See Assigning text lists and graphic lists (Page 7440).

User-defined columns are excluded from this.

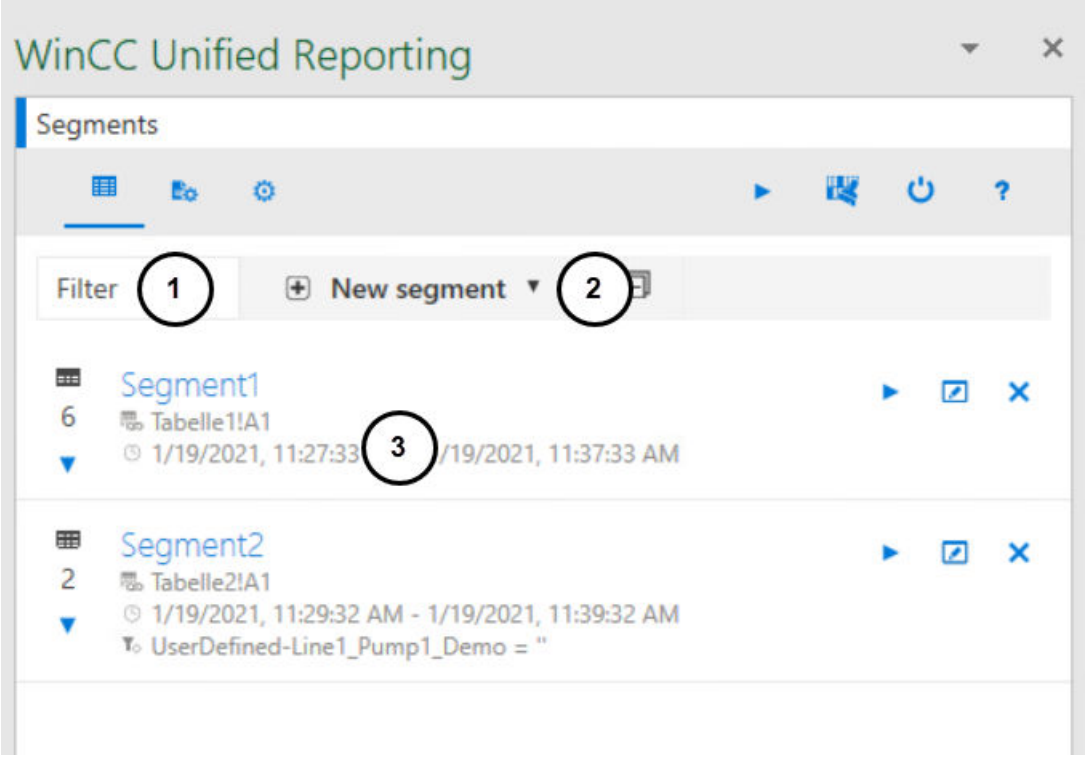
See also

Basic information on segments (Page 7404)

The segment user interface

Structure

The interface for creating and editing segments has the following structure:



① Filter
Filters the list of segments by name.

② Button for creating a segment

③ List of segments
Each segment has buttons for reading in, editing and deleting the segment.
The following configuration is displayed for each segment:

- Segment name
- Number of data source items
- Insertion location of the segment in the Excel file
- Time span
- If context filters have been configured: The filter string

A click on the segment opens the area with the data source items.

Create segments

Requirement

- The "WinCC Unified" tab is visible in Excel.
- The data source is set up.
- To filter the time interval of the time series segment depending on the context: There are contexts in the project that run on the connected Runtime server or are the basis of the configuration file.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "New segment".
3. Select "New time series segment" or "New single value segment".
4. Enter a segment name.

Note

Note the Excel restrictions for naming tables (for example, do not use blanks).

Change the segment name only via the add-in, not via the Excel property "Table name".

Do not change the name of the worksheet after creating the segment. The add-in addresses the segment by the segment name and the worksheet name.

5. For a time series segment, make the following settings in addition:
   - Under "File location" you determine where the segment will be inserted in the file. Enter the name of the worksheet and the cell.
     Alternatively, click "Select a cell" and use the cell currently selected in the Excel file:

     
   - Under "Start" and "End", you determine the time period for which values are read into the segment.

	"Absolute date/ time"	Select a date and a time. The information is absolute to the current date.
	"Relative date/ time"	Select a reference time and a time interval. The information is relative to the current date. See also Formats for relative time information (Page 7413).
	"Date/Time of the cell"	Applies the value of the cell currently highlighted in the Excel file. Make sure that the cell supplies a valid time.
	"Date/Time of the tag"	Applies the value of the set tag. Make sure that the tag supplies a valid time. Possible data types: • DateTime • String • Integer

– (Optional) Under "Properties of the legend table", you configure the contents to be displayed in the table header row of the segment:

"Name" "Start" "End" "State"	General information on the segment
"Context filter"	If the segment time was limited by a context filter: The filter string is output. See step 6.
"Audit status"	If the segment has an audit data source item, the field shows the overall status of the audit data: • Green field: No manipulations of the Audit Trail were found in the queried time range. • Red field: Manipulations of the Audit Trail were found in the queried time range. Single or multiple entries have been deleted, added or changed.
"Header"	The table header row includes a list of the segment's data source items showing general information about these data source items. The information displayed for the data source items depends on their type. Example of contexts: Display name of the context, context provider, hierarchy path, short name of the context, full name of the context, option

Use the check boxes to remove information from or add information to the legend table. To change the sorting in the table header row, move the mouse pointer to a row and shift it using the arrow buttons or drag-and-drop.

6. (Optional) You can filter the time interval of the time series segment depending on the context. You can define up to two filter conditions.
   Proceed as follows:
   - Under "Context filter", click "+" or "Add new condition row".
     The condition line is added.
   - Click on "+" in the condition line.
   - Under "Select context", select the root of the common plant model.
     In the next row, you see the top level of the common plant model.
   - Navigate through the common plant model to plant objects with contexts.
     Plant objects and contexts can be recognized by the following icons:

	Plant object
	Context

   - Select a context.
   - Select an operator.
   - Enter a value.
   - (Optional) Use "+" or "Add new condition row" to create a second condition and select whether the two conditions are to be linked with a logical AND or OR.
7. (Optional) Under "Autofit", configure whether the column width and row height of the data table is automatically adapted to the text read from Runtime.
8. Confirm your entries with "OK".

Result

The segment is created and added to the list of segments:

Next, add data source items to the segment. Your procedure depends on the type of the new data source item.

See also

Tips on design and layout (Page 7449)

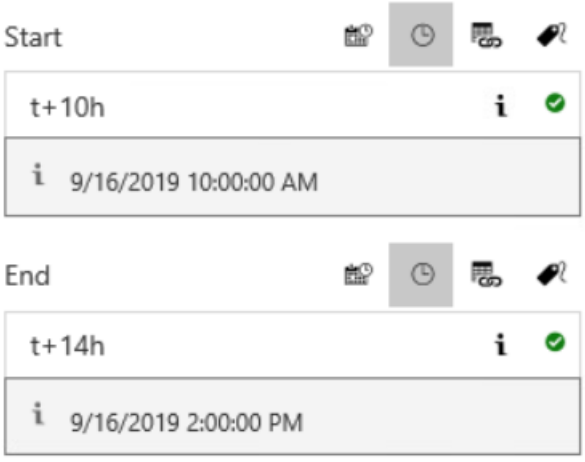
Adding data source items (Page 7415)

Working with configurations (Page 7427)

Formats for relative time information

Definition of a relative time information

The relative times are entered using a reference time and a time interval.



Reference time

Use one of the following characters for the reference time:

- "*" - Now
- "t" (today) - Today at midnight
- "y" (yesterday) - Yesterday at midnight
- "1-31" - Specific day of the current month

Time interval

- "y" (year): +1y = plus 1 year
- "mo" (month): +1mo = plus 1 month
- "w" (week): +1w = plus 1 week
- "d" (day): +1d = plus 1 day
- "h" (hour): +1h = plus 1 hour
- "m" (minute): +1m = plus 1 minute
- "s" (second): +1s = plus 1 second
- "ms" (milliseconds): +250ms = plus 250 milliseconds

Examples

- *-1y: One year ago today
- t+8h: Today at 8:00 am
- y+8h: Yesterday at 8:00 am
- 1+8h: The first day of the current month at 8:00 am
- *-1d: One day ago
- *-2h-30m-30s: 2 hours, 30 minutes and 30 seconds ago

See also

Create segments (Page 7409)

Edit segments

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Edit" next to a segment in the list of segments.
3. Edit the segment.
   You can make the same settings as when creating the segment.

Delete segments

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Delete" next to a segment in the list of segments.
3. Confirm your entries with "OK".

Adding data source items

Adding log alarms

Requirement

- There are logging alarms in the project that runs on the connected Runtime server or is the basis of the configuration file.
- The "Alarm" option is enabled in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Adding logging alarms

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded into the add-in.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Alarms" option.
5. Select the "Alarm" entry under "Select alarms".
6. Under "Select alarms", select the entry "Alarm [ID]".

   Note

   Change selection criteria

   After you have added alarms, you can change the selection criteria and add more data source items to the segment.

   For example: Output tags and alarms in the same segment.

7. To cancel your selection, select the entry "Alarm [ID]" under "Selected data source items" and click "Delete".
8. Confirm with "OK".

Result

- The data source item for logging alarms is added to the add-in below the segment.
- The configuration of the data source item controls which data is added when importing the runtime data into the data table.

  Note

  With the default setting, the data source item uses the default configuration. It shows all logging alarms of the project.

To display data that deviates from the default configuration, select one of the following options:

- Select a different matching configuration.
- Create your own configuration.
- Edit a configuration.
- Overwrite a configuration locally.

See also

Creating or editing configurations for log alarms (Page 7427)

Select configuration (Page 7438)

Working with configurations (Page 7427)

Adding alarm statistics

Introduction

To output statistical calculations for logging alarms in a report, add alarm statistics to a report template. The following calculations are available:

- Frequency of an alarm
- Average display time per state machine
- Total display time per state machine
- Maximum display time per state machine
- Minimum display time per state machine

The alarm statistics add columns with statistical calculations and columns with general alarm properties of the recorded alarms to the reports.

You can find more information about calculations in alarm statistics in the help for the alarm control.

Requirement

- The "Alarm" option is enabled in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a time series segment.
   The segment is extended by the area for the data source items.

3. Click "+".
4. Select the "Alarm" option.
5. Under "Select alarms", select the entry "Alarm statistic [ID]".
6. Under "Select alarm statistic" select the entry "Alarm statistic [ID]".

   Note

   Change selection criteria

   After adding the alarm statistics, you can change the selection criteria and add more data source items.

7. (Optional) To cancel your selection, select the entry "Alarm statistic [ID]" under "Selected data source items" and click "Delete".
8. Confirm with "OK".

Result

The added data source item for alarm statistics is displayed below the segment and inserted into the data table.

First, the data table shows the contents configured in the default configuration for alarm statistics. To output other contents, select or create a configuration.

Add logging tags

Requirement

- The project on which the connected Runtime server runs or the basis of the configuration file has logging tags.
- The "Logging tag" option was selected while setting up of the connection.
- The "WinCC Unified" tab is visible in Excel.
- A single value segment or time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Logging tag" option.

5. Optional: To reduce the load time, filter which tags are loaded to the selection under "Add filter".
   The preset filters "*" return all logging tags of the project.
   - "Tag name": Enter the name of the tag whose logging tags you want to add.
   - "Logging tag name": Enter the name of the logging tags you want to add.

   Note that the entry is case-sensitive.

   Note

   Filter by partial string

   You use the wildcard "*" to filter by partial strings.

   For example:
   - *T* returns all tags with a "T" in their name.
   - *T returns all tags that end in "T".
   - T* returns all tags that start with "T".

   When filtering for structures, the separators must be part of the filter string.

   For example: The following filters return the logging tags for all tags of the device "HMI_RT_1":
   - Filter for tag: "HMI_RT_1::*"
   - Filter for logging tag: "*"

6. Click "Load".
   The logging tags of the project are filtered and provided under "Select tags".
7. Optional: Further reduce the number of tags that are offered for selection by clicking next to "Select logging tags" and entering another filter string.
   The list of tags you are being offered is filtered while you type.
8. Select one or more tags under "Select logging tags".
   The tags are added to the "Selected data source items" list.

   Note

   Change selection criteria

   After you have added a tag, you can select a different option or a different filter and add additional data source items.

   For example: Output KPIs and logging tags in the same segment.

9. To remove one or more data source items from "Selected data source items", select them and click "Delete".
10. Confirm with "OK".
    The added logging tags are shown below the segment and added to the Excel table.
11. If you have added the logging tag to a single value segment:
    - In the Excel worksheet, select the cell in which the logging tag is to be inserted.
    - Click the "Select a cell" button on the data source item of the logging tag.
      Alternatively, enter the name of the worksheet and the cell.

See also

Create or edit configurations for logging tags (Page 7429)

Working with configurations (Page 7427)

Adding tags

Requirement

- The project on which the connected Runtime server runs or the basis of the configuration file has tags.
- The "Tag" option was enabled when the connection was set up.
- The "WinCC Unified" tab is visible in Excel.
- A single value segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select the single value segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Tag" option.
5. Optional: To reduce the load time, filter which tags are loaded to the selection under "Add filter".
   Under "Tag name", enter a filter, e.g. the name of the tag. Note that the entry is case-sensitive. The default filter "*" returns all tags of the project.

   Note

   Filter by partial string

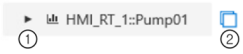
   You use the wildcard "*" to filter by partial strings.

   For example:
   - *T* returns all tags with a "T" in their name.
   - *T returns all tags that end in "T".
   - T* returns all tags that start with "T".

   When filtering for structures, the separators must be part of the filter string.

   For example: The filter "HMI_RT_1::*" returns all tags of the device "HMI_RT_1".

6. Click "Load".
   The tags of the project are filtered and provided under "Select tags".
   You can recognize structs and arrays in the list by the following items:



① Button to display the members of the struct or array
② "Select all included data source items"
Button that adds all members with a simple data type to the list of selected data source items

7. Optional: Further reduce the number of tags that are offered for selection by clicking next to "Select tags" and entering another filter string.

   The list of tags you are being offered is filtered while you type.

8. Select which tags will be added to the segment. You have the following options:

Target	Procedure	Result
Show the members of a struct or array.	Click the button with the arrow next to the struct or array.	A second "Select tags" list is added, in which you can see all the members of the struct or array. You can add to the segment any members that have a simple data type, e.g. bool, float or string.
Add all members of a struct or array.	Next to the struct or array, click "Select all included data source items".	All members with a simple data type are added to the "Selected data source items" list and marked as selected under "Select tags":
Select tags with simple data type.	Under "Select tag", click the required tags.	The tags are added to the "Selected data source items" list and marked as selected under "Select tags": 

Note

Automatic filtering when displaying the members or selection of all members

If you click the button to display the members of a struct or array or activate the option to select their members, the struct or array is set as a filter:

- The list under "Select tags" only shows the struct or array.
- A second "Select tags" list is added below this, in which you can see all members of the struct or array.

To see all available tags again, delete the filters.

Note

Change selection criteria

After you have added a tag, you can select a different option or a different filter and add additional data source items.

9. To remove tags from the segment, click on the tags in "Selected data source items" and click "Delete".
10. Confirm with "OK".

The added tags are added to the segment.

When the report template is updated in the add-in and when the report is generated in runtime, the tag values are inserted into the data table.

See also

Creating or editing configurations for tags: (Page 7432)

Working with configurations (Page 7427)

Adding contexts

Introduction

To display in a report which contexts are to run during a certain time period, add only contexts to a segment in the report template.

To display which process data has been accumulated during the runtimes of a context, add the context and other data source items, such as logging tags or logging alarms, to the segment.

Requirement

- There are contexts in the project that run on the connected Runtime server or are the basis of the configuration file.
- The "Context" option is enabled in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Adding a context to a segment

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Context" option.

5. Select a context:
   - Under "Select a context definition", select the root of the plant model.
     In the next row, you see the top level of the common plant model.
   - Navigate through the common plant model to plant objects with contexts.
     Plant objects and contexts can be recognized by the following icons:

	Plant object
	Context

   - Select the desired contexts.
     All selected contexts are included in the "Selected data source items" list

Note

Change selection criteria

After you have added a context, you can select a different option and add additional data source items.

For example: Context and logging tags in the same segment.

6. To remove one or more data source items from "Selected data source items", select them and click "Delete".
7. Confirm with "OK".

Result

The selected contexts are displayed below the segment and inserted into the data table.

If you do not want a context to use the default configuration, select its configuration next.

Content of the data table after executing the segment

In segments to which only contexts or contexts and user-defined columns have been added:

- A line is inserted for each context whose runtime falls within the time period of the segment.
- "Time stamp" column: The time at which the context was started

In segments that combine contexts with logging tags or logging alarms:

- All logged values with the same time stamp are displayed per row.
- "Time stamp" column: The logging event
- "Start time" column: The time at which the context was started
- "Context " <Context name>"" column: The value passed to the context at start
- If no context was started at the time of logging, the context cells remain empty.

Example

The following data source items were added to a segment:

- The "Product" context
  Runtime of the context: 15:00:00 to 19:59:59 hours
  The context was started with the "Orange lemonade" value.
- The "Logged_Rotation" logging tag
  Logging cycle: 2s
- The "Logged_Temperature" logging tag
  Logging cycle: 5s
- The user-defined "Unit" column
  It contains the unit for "Logged temperatures".

Content of the data table after execution of the segment:

	A	B	C	D	E	F	G
1	Time stamp	Context "Product"	.../Line1-Product:StartTime	.../Line1-Product:EndTime	Logged_Rotation	Logged_Temperature	Unit
2	Mo, 24 Feb 2020 15:00:00,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	10	20	°C
3	Mo, 24 Feb 2020 15:00:02,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	100		°C
4	Mo, 24 Feb 2020 15:00:04,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	500		
5	Mo, 24 Feb 2020 15:00:05,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99		40	°C
6	Mo, 24 Feb 2020 15:00:06,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	750		°C
7	...	...	...	...	...	...	...
8	Mo, 24 Feb 2020 20:00:00,100				650		°C

Lines 2 to 6	Values were logged for "Logged_Rotation" and "Logged_Temperature", while the "Product" context ran with the "Orange lemonade" value.
Line 8	A value was logged for "Logged_Rotation" while no context was running.

See also

Contexts (Page 7240)

Adding user-defined columns

Introduction

User-defined columns supplement the data of the other data source items of a time series segment with additional information:

- With a fixed string
  The string appears in each cell of the column.
  Example: Display measurement unit of the tag values in report
- With a formula
  The formula is calculated during generation for each cell in the dynamic column.
  Example: The sum of the tag values output in the report.

The configuration of the user-defined column controls which string or formula it uses.

Requirement

- The "User-defined column" option was enabled when the connection was set up.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the option "User-defined column".
5. Enter the name of the column under "name".
6. Click "Select" or press <ENTER>.
   The column is included in the list "Selected data source items".

   Note

   Change selection criteria

   After you have added a column, you can select a different option or a different filter and add additional data source items.

7. Select a configuration for the user-defined column.
8. To remove one or more data source items from "Selected data source items", select them and click "Delete".
9. Confirm with "OK".

The added columns are displayed below the segment and inserted into the data table.

See also

Creating and editing configurations for user-defined columns (Page 7433)

Select configuration (Page 7438)

Working with configurations (Page 7427)

Add Audit

Introduction

To output the Runtime device Audit Trail in a report, add an Audit data source item to a report template.

You can find more information about the Audit option in WinCC Unified in the TIA Portal help.

Requirement

- The Audit option was activated in the engineering for the Runtime device.
- The "Audit" option is activated in the connection settings of the Excel add-in.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a time series segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Audit" option.
5. Select the Audit Trail.
6. (Optional) To undo your selection, select the Audit Trail under "Selected data source items" and click "Delete".
7. Confirm with "OK".

Result

The Audit data source item is displayed below the segment.

If an Audit Trail is configured for the data source, the Audit data is inserted into the report when the Runtime data is read into Excel and when it is generated in Runtime:

- In the legend table: Identifier of the overall status of the Audit Trail for the queried time range in the "Audit Status" field

Value	Description
Green	No manipulations of the Audit Trail were found in the queried time range.
Red	Manipulations of the Audit Trail were found in the queried time range. Single or multiple entries have been deleted, added or changed.

Requirement: The "Audit status" option is activated on the segment under "Header properties".

Note

Overall status for check mode "None"

If the check mode "None" is set in the configuration of the audit data sources item, the "Audit status" field is always green.

- In the data table of the segment: Identifier of manipulations

Type of manipulation	Identifier in the data table
Value of entries changed	Directly at the entries
Entries added	
Entries deleted	The manipulated time range receives a start and end entry.

First, the data table shows the contents configured in the standard configuration for Audit. To output other contents, select or create a configuration.

Delete data source elements

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source element is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Expand a segment by clicking on it.
   The area for adding and editing data source elements appears.
3. Move the mouse pointer over a data source element and click "Delete".

Working with configurations

Basics of configuration

The *configuration* of a data source item defines the values of a data source element that are displayed in a segment or how they are calculated and displayed.

There are specific configuration settings for each data-source-item type.

Data source items used in time series segments use a different configuration than data source items used in single-value segments.

You have the following options:

- Use standard configuration.
  There is a standard configuration for all types of data source items. Once added, data source items use the default configuration of their type.
  You can edit the standard configurations.
- Use user-defined configuration.
  You can create any number of user-defined configurations for all types of data source items.
  You can select one of the user-defined configurations on the data source item.
- Overwrite a configuration locally.
  You can overwrite the configuration selected at the data source item locally.

Creating or editing configurations for log alarms

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > Logging alarm configuration".
4. Enter the name of the configuration under "Configuration name".
5. (Optional) Enter texts or graphics from a text list or graphic list in the standard column instead of the alarm IDs.
   See Assigning text lists and graphic lists (Page 7440).
6. (Optional) Change the default settings of the optional columns. The optional columns are used to display the alarm properties.
   See Configuring optional columns (Page 7439).

7. (Optional) Filter the logging alarms to be displayed. You define a filter query for this purpose. The filter query can consist of up to two conditions.
   Proceed as follows:
   - Under "Filter", click "+" or "Add new condition row".
   - Select an alarm property, an operator, and enter a value.
   - Optional: Use "+" or "Add new condition row" to create additional conditions. Select whether the conditions are to be linked with a logical AND or OR.
8. Enable the option "Use system colors" so that the alarms are highlighted with the same colors as in the alarm control.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7443).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for logging alarms.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Creating or editing configurations for an alarm statistics

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > Alarm statistics configuration".

4. Enter the name of the configuration under "Configuration name".
5. (Optional) Enter texts or graphics from a text list or graphic list in the standard column instead of the alarm IDs.
   See Assigning text lists and graphic lists (Page 7440).
6. (Optional) Change the default settings of the optional columns. The optional columns are used to display the statistical calculations and alarm properties.
   See Configuring optional columns (Page 7439).
7. (Optional) Filter the contents to displayed in the alarm statistics. You define a filter query for this purpose. The filter query can consist of up to two conditions.
   Proceed as follows:
   - Under "Filter", click "+" or "Add new condition row".
   - Select an alarm property, an operator, and enter a value.
   - Optional: Use "+" or "Add new condition row" to create additional conditions. Select whether the conditions are to be linked with a logical AND or OR.
8. Enable the option "Use system colors" so that the alarms are highlighted with the same colors as in the alarm control.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item.

See Setting a display name for standard column (Page 7443).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for alarm statistics.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Create or edit configurations for logging tags

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration".
4. To create a configuration for logging tags in a time series segment, select the entry "Logging tag configuration".
   To create a configuration for logging tags in a single value segment, select the entry "Single value configuration logging tag".
5. Enter the name of the configuration under "Configuration name".
6. Under "Calculation mode", select the data to be written if no current value is available.
7. (Optional) If the configuration is for logging tags with the numeric data type, you can output texts or graphics from a text list or graphic list in the standard column instead of the tag value. See Assigning text lists and graphic lists (Page 7440).
8. Set the other settings as described below.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7443).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for logging tags.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Additional settings for time series segments

In time series segments, the following additional settings are available for logging tags:

Setting	Description
"Interval"	Only for the "Stepped" and "Interpolated" calculation modes.
"Columns" > "Quality Code"	(Optional) Change the default settings of the optional "Quality Code" column. See Configuring optional columns (Page 7439).

Additional settings for single value segments

In single value segments, the following additional settings are available for logging tags:

Setting	Description
"Time stamp"	Determine the date and time for which the value is read. Proceed as described below.
"Show captions"	Define whether a header is displayed in the columns for the time stamp, the data source item and the quality code.
"Show time stamp"	Determine whether and where this information is displayed in the table. The information is always in relation to the value cell.
"Show data source item"	
"Show quality code"	

To set the "Time stamp", select one of the following options:

	Absolute time information	Select a date and a time. The information is absolute.
	Relative time information	Select a reference time and a time interval. The information is relative to the current date.
	Read time information from cell	Applies the value of the cell currently highlighted in the Excel file. Make sure that the cell supplies a valid time.
	Read time information from tag	Applies the value of the set tag. Make sure that the tag supplies a valid time. Possible data types: • DateTime • String • Integer

See also

Calculation modes for data source elements (Page 7447)

Creating or editing configurations for tags:

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration": 
3. Click "New Configuration> Tag single value configuration".
4. Enter the name of the configuration under "Name".
5. (Optional) If the configuration is for tags with the numeric data type, you can output texts or graphics from a text list or graphic list in the standard column instead of the tag value. See Assigning text lists and graphic lists (Page 7440).
6. Set the other settings as described below.
7. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7443).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration".
3. Click a configuration for tags.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Settings for single value segments

In single value segments, the following settings are available for tags:

Setting	Description
"Show captions"	Select whether a header is displayed in the columns for the time stamp, the data source item and the quality code.
"Show time stamp"	Select whether the time stamp is output with the value.
"Show data source item"	Select whether the quality code is output with the value.
"Show quality code"	Select whether the quality code is output with the value.

Creating or editing configurations for contexts

Requirement

- The "WinCC Unified" tab is visible in Excel.

Core statement

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration".
4. Select the entry "Configuration context".
5. Enter the name of the configuration under "Configuration name".
6. (Optional) Change the default settings of the optional columns. The optional columns are used to display important contextual information.
See Configuring optional columns (Page 7439).
7. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7443).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for contexts.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Creating and editing configurations for user-defined columns

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > User-defined column configuration".
4. Enter the name of the configuration under "Configuration name".
5. Under "Formula", select one of the following options:
   - Enter a fixed string.
     The string is transferred into each cell of the column.
   - Enter an Excel formula.
     The formula is copied into each cell of the user-defined column and adapted to the respective row.
     To prevent a part of the formula from being adjusted, place the character "$" in front of it.
     Example

Formula in configuration		=B2+C2	=B$2+C2
Adapting the formula in the report	in line 2	=B2+C2	=B2+C2
	in line 3	=B3+C3	=B2+C3
	in line 4	=B4+C4	=B2+C4

   Note

   No validity check

   The formula is not tested for correctness during either input or dynamic adaptation.

6. Confirm your entries with "OK".

Adding or editing configurations for audit

Introduction

Check mode

The check mode of the configuration of an audit data source item determines

- Whether an integrity check is performed when the Runtime data is read, and what is checked. You can output the overall result of the check in the table header row in the "Audit status" field.
- Which audit data records are provided in the data table.

Possible check modes:

"None"	Provides the data for all audit data records that fall within the requested time range. No integrity check is performed. Default setting
"Check only"	Checks all audit data records that fall within the requested time range without providing their data. It is tested whether data records have been manipulated, deleted or added.
"Check entries"	Check the audit data records and provides their data that fall within the queried time range and that have not been deleted from the Audit Trail or subsequently added. It is checked whether data records have been manipulated.
"Check all"	Checks all audit data records and provides their data that fall within the queried time range. It is tested whether data records were manipulated, deleted from the audit trail or subsequently added.

Filter type

An Audit data record consists of two entries:

- An entry for the user expectation
- An entry for the system response.

User expectation and system response may differ. In addition, there are situations in which only one of the two data records is created.

The filter type controls which data records and which entries are included in the report.

Possible filter types:

Filter type	User expectation equals system response	User expectation does not equal system response	Data record entry for user expectation or system response is missing
"Show all data in detail"	Both data record entries are inserted.		The existing data record entry is inserted.
"Show data and conformity errors"	The data record entry with the user expectation is inserted.	Both data record entries are inserted.	
"Show only data with conformity errors"	No data record entry inserted.		

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > Audit configuration".
4. Enter the name of the configuration under "Name".

5. Select a check mode:
6. Specify a filter type.
   Preset value: "Show data and conformity errors"
7. (Optional) Change the default settings of the optional columns. The optional columns are used to display the audit attributes.
   See Configuring optional columns (Page 7439).
8. (Optional) To further filter the inserted content, define a filter query.
   The filter query can consist of up to two conditions. Proceed as follows:
   - Under "Filter", click "+" or "Add new condition row".
   - Select an Audit attribute, an operator and enter the value of the attribute.
   - Optional: Use "+" or "Add new condition row" to create additional conditions. Select whether the conditions are to be linked with a logical AND or OR.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7443).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

   
3. Click a configuration for Audit.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Examples of the configuration of the filter type

The following table contains examples of data records that were generated in Runtime through changes to tags monitored by Audit:

Data record ID	Tag name	Modified by	Old value	New value	Description
1A	Motor1_Speed	User1	0	10	An operator changes the speed of a motor in an I/O field of an HMI screen.
1B	Motor1_Speed	System	0	10	User expectation and system response are identical.

Data record ID	Tag name	Modified by	Old value	New value	Description
2A	ValvePercen-tile	User1	0	100	An operator opens a valve using a slider on an HMI screen. The valve has a physical blockage and cannot be opened. Therefore, no data record entry for the system response is generated.
3A	ValvePercen-tile	User1	0	99	A physical block has been removed and the operator repeats the entry. The valve reacts, but cannot be fully opened. User expectation and system response differ.
3B	ValvePercen-tile	System	0	49	
4B	Mo-tor2_Speed	System	0	20	An operator changed the speed of another motor. The resulting data record was manipulated, and the user expectation entry was deleted. There is only one entry for the system response.

The following table shows which data record entries are inserted into the data table depending on the filter type selected when generating the report:

Data record ID	Tag name	Modified by	Old value	New value
Filter type "Show all data in detail"				
1A	Motor1_Speed	User1	0	10
1B	Motor1_Speed	System	0	10
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20
Filter type "Show data and conformity errors"				
1A	Motor1_Speed	User1	0	10
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20
Filter type "Show only data with conformity errors"				
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20

Select configuration

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source item is available.
- There is a user-defined configuration for the type of the data source item.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The segment is extended by the area for the data source items.
3. Select the desired configuration from a data source item in the drop-down list.
4. Click "OK".

Result

The changes are applied the next time you read in the runtime data.

Overwrite a configuration locally

A local configuration overwrites the configuration selected at the data source item. It applies only to the data source item for which it was entered.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source item is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The segment is expanded to include the plant complex for the data source items.
3. Move the mouse over a data source item and click "Edit".
   You create a local configuration that first adopts the values of the original configuration.
4. Enter a name for the local configuration.
5. (Optional) Set a display name. See Setting a display name for standard column (Page 7443).
6. Make the remaining settings as required.
   You can make the same settings as in the default and custom configurations.
7. Confirm your entries with "OK".

Result

The changes are applied the next time you read in the Runtime data.

Delete configuration

Requirement

A configuration is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration".
3. Move the mouse to a configuration.

   Note
   Default configurations cannot be deleted
   You can edit default configurations but not delete them.

4. Click "Delete".

Result

- The configuration is deleted.
- Data source items with this configuration obtain a local configuration with the same settings.

Configuring optional columns

Introduction

In time series segments, data source items of the following types have optional columns:

- Logging tag
- Logging alarm
- Alarm statistics
- Audit
- Context

The optional columns of a data source item depend on its type. The configuration of the data source items controls whether and how the data table shows these columns.

This section describes how to configure the optional columns.

Requirements

The data source item configuration is open. The configuration must apply to a time series segment.

Showing and hiding columns

1. To show an optional column in the data table, enable the option for the desired column in the "Columns" area.
2. To hide a column, disable its option.

Changing the column title

The data table uses as default column titles the identifiers you see in the "Columns" area. To change the default column titles, do the following.

1. In the "Columns" area, move the mouse pointer to an optional column.
2. Click the button with the pin.
3. Assign a unique column title.

Note

Localization

The column title is stored in the Runtime language currently set in the basic settings of the add-in.

To localize the column title, change the Runtime language and repeat your entry in the new language.

Changing the column sequence

To change the order of the optional columns in the data table, proceed as described in Changing the column sequence (Page 7444).

Assigning text list or graphic list

The values of numeric columns can be replaced by texts or graphics when the Runtime data is read in.

To assign a suitable text list or graphic list to the property, proceed as described in Assigning text lists and graphic lists (Page 7440).

Assigning text lists and graphic lists

Introduction

If standard columns and optional columns of data source items output numerical values, you can assign text lists and graphic lists to these columns. When the Runtime data is read in, the cell values of these columns are replaced by texts or graphics from the assigned lists.

This function improves the readability of the report and helps to draw the reader's attention to important information.

Note

Restrictions

- Tags/logging tags
  Assign a text list or graphic list to the standard column of data source items with a Tag or Logging tag type only if the tag or logging tag has a numeric data type.
  You can assign a text list or graphic list to the optional "Quality Code" column regardless of the data type of the tag.
- User-defined columns
  It is not possible to assign a text list or graphic list for data source items with the User-defined column type.
- Context and audit
  Usually, the names of context objects and audit objects displayed in the standard column do not contain purely numerical values. It is not recommended to assign a text list or graphic list.

Example

Add two data source items with the same logging tag to a time series segment.

For the first data source item, use the default configuration. This causes the report to output the tag value in the standard column.

For the second data source item, select a configuration in which a graphic list is assigned to the standard column. The graphic list contains representational graphics staggered by value range. As a result, the report outputs a graphic in the standard column.

After reading in the Runtime data, the standard column of the second data source item makes readers of the report aware of limit violations. Readers can get the exact tag value from the standard column of the first data source item.

Requirements

- A segment with a data source item was created in the add-in.
- Suitable text lists or graphic lists have been configured in engineering for the connected data source.

Assigning text lists and graphic lists to the standard column

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The data source items of the segment are displayed.
3. Move the mouse over a data source item and click "Edit".
   The local configuration of the data source item opens.

4. Select a suitable list under "Assign text/graphic list".
5. To preview the selected list and its graphics or texts, click the "i" button.
   To hide the preview, click the "i" button again.

Assigning optional columns to text lists and graphic lists

1. Click on "Segments" in the "Configuration" group.
2. Select one of the following options:
   To make the assignment apply to a specific data source item, follow these steps:
   - Select the segment.
     The data source items of the segment are displayed.
   - Move the mouse over the data source item and click "Edit".
     The local configuration of the data source item opens.

   To make the assignment apply to multiple data source items of the same type, follow these steps:
   - Click "Data source item configuration": 
     You can see all default and custom configurations.
   - Click on the desired configuration.
     The configuration opens.
3. In the "Columns" area, click the following button next to the desired optional column:

   An interface for assigning a text list or graphic list is loaded into the add-in.
4. Select the desired graphic list or text list from the drop-down list.
5. To preview the selected list and its graphics or texts, click the "i" button.
   To hide the preview, click the "i" button again.

   Note

   If you are connected offline to the data source, no preview of graphic lists is available.

6. Confirm your entries.

Result

When the Runtime data is read in, the assigned lists are searched for an entry that matches the actual cell value:

- If a matching entry is found, the corresponding text or graphic is inserted into the data table.
- If no matching entry is found, the actual cell value is inserted.

Note

The assignment of graphic lists slows down the import of the Runtime data into the Excel add-in.

See also

Standard column (Page 7406)

Basic information on segments (Page 7404)

Setting a display name for standard column

Introduction

You can assign a display name for the standard column of a data source item. When a display name is set, it is used in the data table instead of the default column title and is listed in the table header row.

Display names make reports clearer, for example, when data source items for contexts or tags have very long names.

You set the display name in the local configuration of the data source item.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- There is a segment with a matching data source item.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Expand a segment by clicking on it.
   The area for adding and editing data source elements appears.
3. Move the mouse pointer to the data source item and click "Edit".
   The local configuration of the data source item opens.
4. Enter the desired column title in "Display name".
   The column title must be unique within the segment.

   Note

   Localization

   The column title is stored in the Runtime language currently set in the basic settings of the add-in.

   To localize the column title, change the Runtime language and repeat your entry in the new language.

5. Confirm your entry with "OK".

Result

- The data table uses the display name as the column title for the standard column of the data source item.
- As long as the following conditions are met, the "Display name" column is inserted into the table header row:
  - Display of the header row in table header row is enabled.
    Make this setting at the segment.
  - A display name is set for at least one data source item of the segment.

  The column lists the display names of all data source items. If no display name is set for a data source item, its cell remains empty.

Note

- If you assign a different configuration to the data source item, the display name is retained.
- To return to the display of the default column title after assigning a display name, enter the name of the data source item under "Display name".

See also

Standard column (Page 7406)

Overwrite a configuration locally (Page 7438)

Changing the column sequence

Introduction

For a time series segment, you can change the default column order of the data table.

You have the following options:

- Specify the order which the data source items have in the data table.
- For each data source item: Set the order of its optional columns.

Note

The time stamp column always appears first.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A time series segment has been created.

Change the order of data source items

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click the time series segment in the list of time series segments.
   The data source items of the segment are displayed.
3. Left-click a data source item and move it up or down using drag-and-drop operation.

Result

The order of data source items in the segment interface is changed.

The next time the Runtime data is read in, the data table outputs the data source items in this order.

Changing the order of optional columns

Procedure

1. Select one of the following options:
   To change the column order of a particular data source item, follow these steps:
   - Select the segment.
     The data source items of the segment are displayed.
   - Move the mouse over the data source item and click "Edit".
     The local configuration of the data source item opens.

   To change the column order for all data source items that use the same configuration, follow these steps:
   - Click "Data source item configuration":

     You can see all default and custom configurations.
   - Click on the desired configuration.
     The configuration opens.
2. Move the mouse pointer to a column under "Columns".
   The columns you see depend on the type of data source item.
3. Move the column up or down using the arrow buttons or drag-and-drop.

Result

The order of the optional columns in the configuration is changed.

The next time the Runtime data is read in, the data table outputs the optional columns in the changed order.

See also

Basic information on segments (Page 7404)

Configuring optional columns (Page 7439)

Reading Runtime data in Excel

Note

Reading in Runtime data in Excel is used for testing. It is not intended for mass retrieval of data, as is the case when report jobs are executed in Runtime.

Requirement

An online connection is established.

Reading in all segments

1. Select "WinCC Unified > Segments".
2. Click "Update all":

Reading in individual segments

1. Select "WinCC Unified > Segments".
2. Next to a segment in the list of segments click, "Update":

Result

The segment or segments are run. The Runtime data of your data source items are read into Excel.

Note

Controlling the column width and row height

When the automatic adjustment of the column width and row height is disabled in the segment properties, the text read in may be truncated or the formula results are replaced with "#" characters.

Check the column widths and row heights and adjust them manually, if required, or select automatic adjustment. Manual adaptations only apply in the Excel add-in. They are not included in the generated reports.

Note

Removing Runtime data from report template

Remove the Runtime data from the report template before you save the report template and make it available for uploading to Runtime.

To do this, click the "Delete Runtime data" button  in the toolbar of the Excel add-in.

Diagnostics during the data query

Successful execution of the data query is documented by the add-in with a status message in the table.

If an error occurs during the data query, a general error message is displayed under status. In addition, detailed error messages are displayed in the "ErrorLog" worksheet.

Calculation modes for data source elements

If there is no current value for a data source item for a requested point in time, the following calculation modes are available.

Calculation modes for tags

The following calculation modes are available for tags of a time series segment:

Calculation mode	Description
Raw	The actual value available for the specified period. If no data are available, no value is output.
Stepped	If no data are available, the last value is used. With this mode you can also use values with an invalid quality code.
Interpolate	The values are interpolated linearly for the specified time period. With this mode you can only use values with a valid quality code.

The following calculation modes are available for tags of a single value segment:

Calculation mode	Description
Interpolate	The values are interpolated linearly for the specified time period. With this mode you can only use values with a valid quality code.
Left	If no data is available, the last value before the specified period is used.
Right	If no data is available, the last value after the specified period is used.

See also

Create or edit configurations for logging tags (Page 7429)

Making general settings

Adapting the work area

Undocking and moving the add-in

To enlarge your working area, you can undock the Excel add-in:

1. Open the drop-down list in the header of the add-in.
2. Click "Move".
3. Move the mouse pointer to the desired location and click the left mouse button.
4. To move the add-in again, keep the left mouse button pressed in the header of the add-in and move the mouse.
5. To dock the add-in again, double-click in the header of the add-in.

Adapting the size of the add-in

1. Open the drop-down list in the header of the add-in.
2. Click "Resize".
3. Move the mouse pointer to the left to make the add-in wider or to the right to make it narrower.
4. Left-click when you have reached the desired size.

Changing the language

Changing the add-in language

The Excel add-in automatically uses the same user interface language as Excel. If you are using a language for Excel that is not included in the Unified options, English is used as the default language.

You can select the language for the contents of the report independently of the interface. To select another language, the language must be configured in Runtime.

Selecting the language for the report

1. Select "WinCC Unified > Segments".
2. Click "Basic settings":

3. Under "Runtime language", select the language of the report content.
4. Under "Query language" you select which language data queries have that require user input, e.g. filter definitions.

Zooming in the add-in

Procedure

To zoom in or out of the display in the add-in, press <CTRL> and move the mouse wheel.

Undo and redo

The Excel functions "Undo" and "Redo" are not available in the add-in.

Tips on design and layout

This section includes tips on the visual design of reports. The apply for:

- Report templates
- Reports that were generated as XLSX file

Note

Deviating PDF results

A PDF report created by LibreOffice can deviate in content or layout from a PDF report generated with Excel, for example, if the report template uses common Excel features that LibreOffice does not support, such as special fonts or chart types.

Arranging segments

Always place the segments of a report template side by side or each in their own worksheet.

Because the data tables of the segments grow dynamically, tables can overlap when segments are placed one below the other. In the add-in, this causes an error of the OfficeExtension.Error class when reading in the Runtime data.

Changing the column sequence

See section Changing the column sequence (Page 7444).

Adapt column width and row height

For each segment of a report template, check whether the column widths and row heights of your data table are wide or high enough for the values to be read. If this is not the case, texts in the generated report are truncated or the formula results are replaced with "#" characters.

To do this, select one of the following options:

- In the properties of the segments, select the options for automatic adjustment of the column width and row height.
- Click "Update all" in the report template.
  Values are imported to Excel from the data source. Check the column widths and row heights and adjust them manually, if required.

Note

The manual adaptations apply only in the Excel add-in. They are not included in the generated reports.

Replacing numerical values

If columns of a data source item output numeric values, you can assign text lists and graphic lists to the columns. When the Runtime data is read in, the cell values of these columns are replaced by texts or graphics from the assigned lists. This improves the readability of the report and helps to draw the reader's attention to important information.

Example: Visualizing limit violations of tags with graphics

See section Assigning text lists and graphic lists (Page 7440).

Preparing imported Runtime data

Adjust the cell formatting of the Runtime data, for example, font, color, alignment, or number format. The rows inserted when reading the Runtime data adopt the formatting.

Add diagrams, pivot tables or formulas that graphically visualize, structure or evaluate the data imported from Runtime.

Note

If you have read Runtime data into the report template for better data preparation, remove it before saving the report template and making it available for upload to Runtime.

To do this, click the "Delete Runtime data" button in the toolbar of the Excel add-in.

Set up page

Use "File > Print > Set up page" to define details for printing the report, for example:

- Alignment of the report (portrait format or landscape format)
- Scaling, for example, to print all columns on one page
- Inserting a user-defined header or footer

The print settings set in the report template are applied in Runtime when a report job is executed for PDF generation.

Renaming worksheets and segments

When you add a segment to a report template in the add-in, a table is created in Excel. The add-in addresses the table by the name of the worksheet and segment.

Do not rename worksheets after adding segments.

Do not change the table name of a segment using the Excel property "Table name". Edit the segment in the add-in and rename it there.

17.3.4.12 Rearranging columns at runtime

Introduction

You can rearrange the table columns in the following table-based controls:

- Alarm control
- Trend companion
- Process control
- Parameter set control
- System diagnostics control

Requirement

Configuration of the control in the engineering requires rearrangement of the columns.

Procedure

To move a column, drag-and-drop its column header onto the column header of another column.

Note

The time column cannot be moved.

Result

The moved column is inserted at the position that the cursor had when the drag-and-drop movement ended.

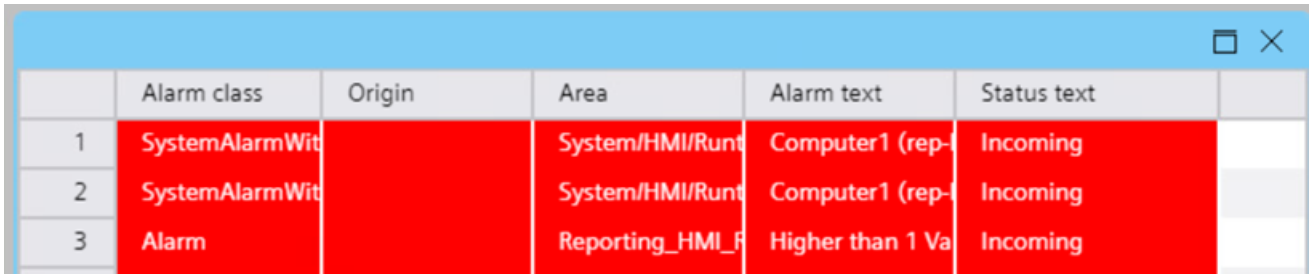
The new arrangement only applies to the current client. If you change the page or refresh the browser window, the arrangement is lost.

Note

If you move a column next to the hidden column and then unhide it, it is always shown to the right of the moved column when it is unhidden.

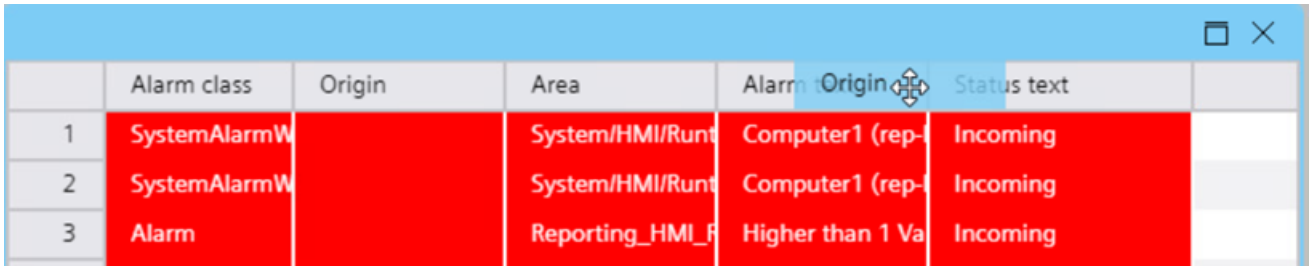
Example 1: Inserting columns to the left or the right

The procedure is illustrated based on the example of an alarm control. In the initial situation, the table of the alarm control has the following column arrangement:

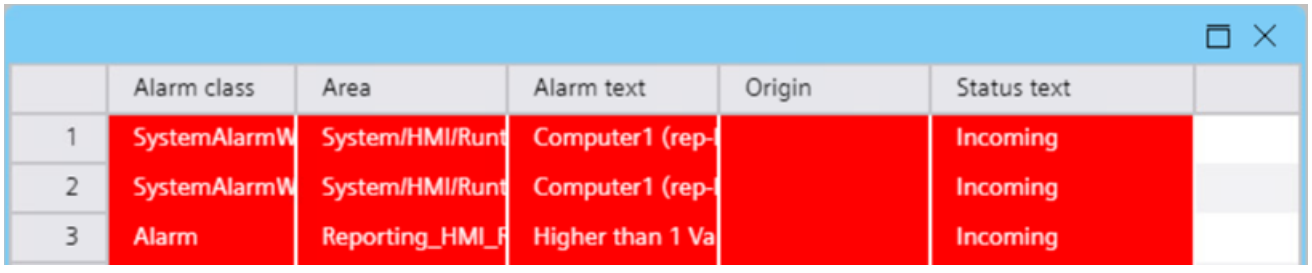


To insert the "Origin" column to the right of the "Alarm text" column, proceed as follows:

1. Use drag-and-drop to move the column header of "Origin" to the right half of the column header of the "Alarm text" column.

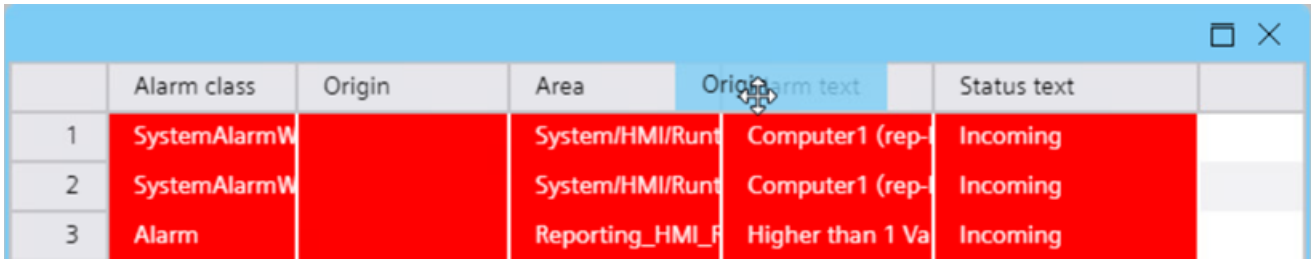


2. The "Origin" column is inserted to the right of the "Alarm text" column.



To insert the "Origin" column to the left of the "Alarm text" column, proceed as follows:

1. Use drag-and-drop to move the column header of "Origin" to the left half of the column header of the "Alarm text" column.



2. The "Origin" column is inserted to the left of the "Alarm text" column.

Example 2: Reordering of columns in combination with hidden columns

The example illustrates the rearrangement of columns in combination with hidden columns.

- The alarm view has the same column order as in Example 1.
- The alarm view has been configured in the engineering system in such a way that the display of the "Origin" column is controlled dynamically in runtime by setting a tag.

To reorder the columns in combination with hidden columns, proceed as follows:

1. Hide the "Origin" column by setting the tag.
2. Insert the "Status text" column to the left of the "Area" column.
3. Unhide the "Origin" column by setting the tag.

The columns have the order "Alarm class", "Status text", "Origin", "Area", "Alarm text".

17.3.4.13 Process diagnostics

Basics of supervision with ProDiag

Introduction

The TIA Portal functionality, ProDiag (Process Diagnostics), is used to monitor and determine errors that occur in your plant or machine. You can use ProDiag to show the type of error, the cause of the error and the location of the error on the HMI device.

Use

You can use ProDiag functions to monitor your plant and to visualize it on an HMI device. The main objective of ProDiag is the reduction of downtime and loss of production after an error occurs, and the avoidance of errors using timely warnings. Diagnostic and display objects provide specific information for the operator for troubleshooting and show the processes on an HMI device on site.

Principle

In STEP 7, you create operand supervisions and configure the settings according to your requirements. When an error occurs, a supervision alarm is generated based on the criteria you have configured. The configured supervision instances are stored in the preset ProDiag function block. You can use the automatically generated ProDiag FBs or create and configure your own ProDiag FBs.

Advantages

ProDiag enables you to configure supervisions and monitor your plant without changing the user program code.

You perform plant diagnostics on your HMI device. The data is automatically synchronized in order to keep the display on your HMI device always up-to-date.

Requirements and licensing

Introduction

You configure the ProDiag supervisions in TIA Portal with STEP 7 and create the screen objects for monitoring and diagnostics with WinCC. You need a license to use the ProDiag functionality and the corresponding screen objects.

Software requirements

To configure ProDiag supervisions, you need the following products:

- TIA Portal STEP 7 Professional
- WinCC Unified

Hardware requirements

ProDiag functionality is available for CPUs of the S7-1500 series with firmware version 2.0 or higher.

The objects for the supervision and diagnostics of plants are available for the following HMI devices:

- WinCC Unified

Note

Objects for supervision and diagnostics of plants can be used under the "Full access" and "Read access" protection levels configured in the CPU.

ProDiag objects under the "HMI access" and "No access" protection levels cannot be used.

Licensing of ProDiag supervisions

The number of ProDiag monitors that you configure with STEP 7 is licensed. You do not need a license for the first 25 supervisions, licenses must be used for additional supervisions.

Number of supervisions	**<= 25**	**<= 250**	**<= 500**	**<= 750**	**<= 1000**	**> 1000**
Number of licenses	None	1	2	3	4	5

Licensing of ProjDiag objects

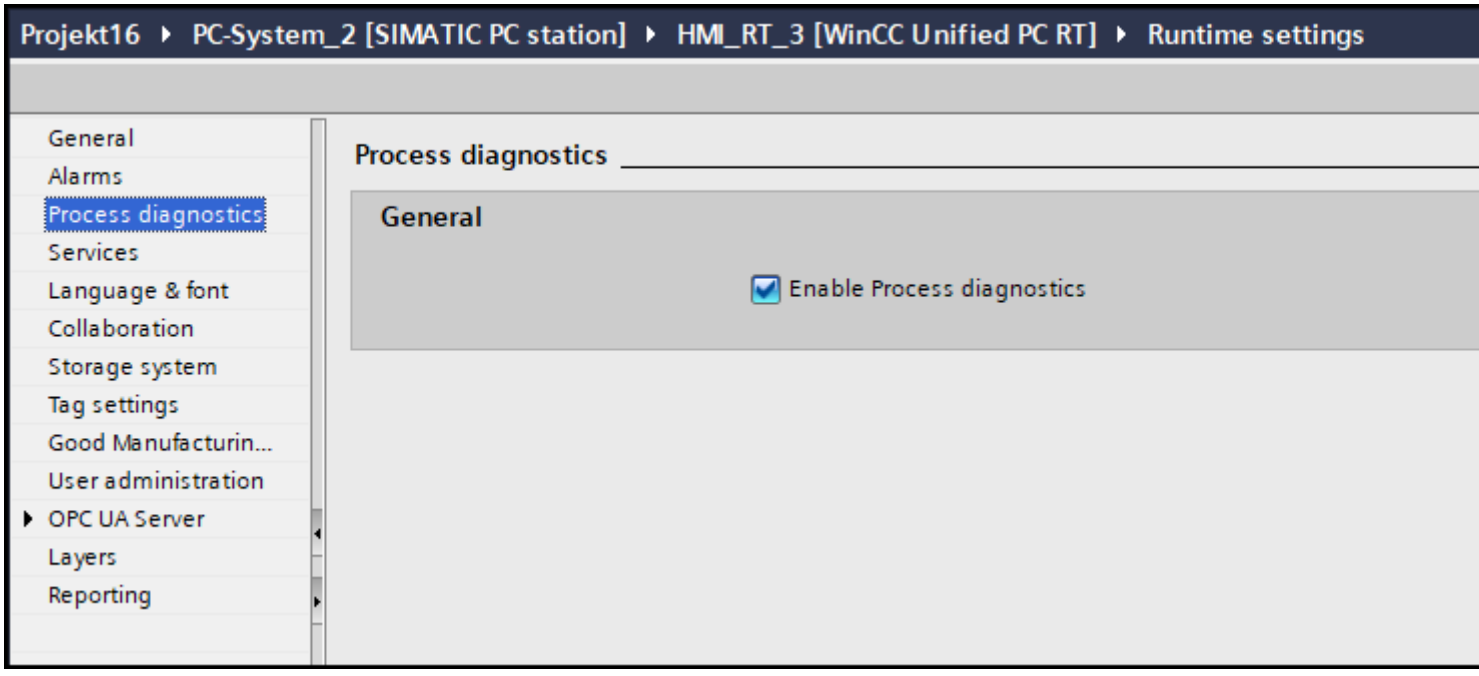
To use the objects for the diagnostics and supervision in conjunction with the ProDiag supervision in your program, you need a ProDiag license, specifically the WinCC Unified Runtime license.

Enable process diagnostics

To activate process diagnostics on an HMI device, follow these steps:

1. Open the "Runtime settings" of the HMI device in the project tree.
2. Under Process diagnostics, select the "Enable process diagnostics" option.

The display of process diagnostic alarms is enabled in runtime.



Objects for the supervision and diagnostics of plants

Introduction

WinCC offers the following objects for displaying the current monitoring status and for fault diagnostics in the program code:

- GRAPH overview
- PLC code view

GRAPH overview

The "GRAPH overview" object is used to display the current program status for executed steps of the GRAPH sequencer.

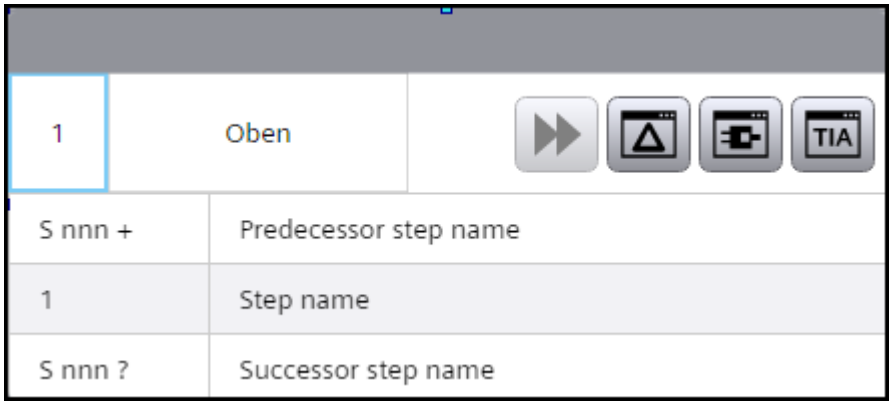
PLC code view

The "PLC code viewer" object is used to display the current program status of user programs that have been programmed in the GRAPH programming language.

GRAPH overview

Use

The "GRAPH Overview" object is used to display the current program status for executed steps of the GRAPH sequencer. Errors during execution of a program are displayed directly at the corresponding step.



The following information is displayed in the "GRAPH Overview" object:

- Name and status of the function block
- Status of initial and simultaneous steps
- Number and name of the first step currently executed step
- Operating mode for running the GRAPH sequencer

WinCC supports the display of step names for the GRAPH blocks in multiple languages starting from Version 6.0. The step names will then be displayed in the selected Runtime language following a language changeover in Runtime. If the selected language is not available in the GRAPH block, the names are displayed in the default language (English).

Note

Device dependency of the "GRAPH Overview" object

The "GRAPH overview" object is available for Unified PC.

Note

Requirement for display in GRAPH overview

For the display of the program status of an S7 GRAPH instance data block in the "GRAPH overview" object to be possible, the instance-specific properties of the block must be set as "Visible in HMI" and "Accessible from HMI".

Layout

In the Inspector window, you customize the position, style, colors and font types of the object. You can adapt the following properties in particular:

- Assigned GRAPH DB tag
- Buttons of the toolbar

Operating mode

Four operating modes are available for running the GRAPH sequence:

- AUTO (default setting) - Automatically switches to the next step when the transition is fulfilled.
- TAP - Automatically switches to the next step when the transition is fulfilled and there is an edge change from "0" to "1" at the T_PUSH parameter.
- TOP - Automatically switches to the next step when the transition is fulfilled or there is an edge change from "0" to "1" at the T_PUSH parameter.
- MAN - The next step is not automatically enabled when the transition is fulfilled. The steps can be selected and deselected manually.

Note

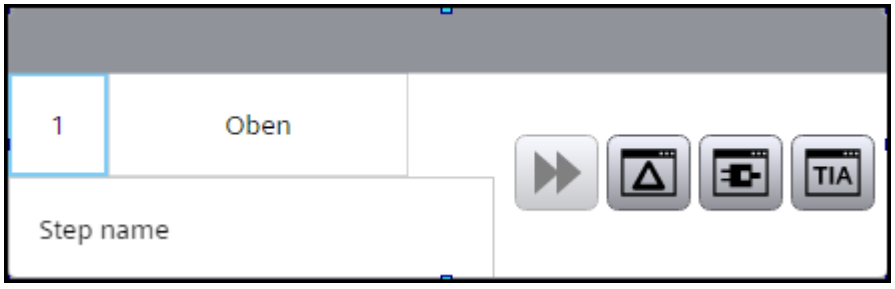
You set the operating mode by modifying the interface parameters of the GRAPH block in your control program.

In WinCC Unified Runtime, you have the option to customize the name for the operating mode that is displayed in the GRAPH overview.

Configuring a compact view

You can also configure a slim GRAPH overview without toolbar buttons and operating mode display.

To display a slim GRAPH overview in single-line compatibility mode, drag the control to the desired size.



Symbols

The symbols displayed in the GRAPH overview are pre-defined:

Symbol	Name	Function
	Error	Indicates that an error has occurred during the execution of a step.
	Initial step	Indicates that the currently executing step is the first step in the GRAPH block.
	Simultaneous step	Shows that there are other simultaneous steps in the GRAPH block in addition to the current one.

Buttons

You specify the buttons that are displayed in the GRAPH overview under "Properties > Miscellaneous > Toolbar > Elements".

Button	Name	Function
	Next Step	Jumps to the next step in parallel step. When you get to the last step, you can jump back to the first step.
	Jump to Alarm Control	Opens the configured alarm view with the error alarm in WinCC Unified. The button is intended to be populated with appropriate system functions/ scripts.
	Jump To PLC Code Viewer	Opens the configured PLC code view. The button is intended to be populated with appropriate system functions/ scripts. Ideally, use the "OpenViewerGraphFromOverview" system function.
	Jump to TIA Portal	Several system functions are available for opening the TIA Portal.

Configuring a GRAPH overview

Introduction

You can use the GRAPH overview to view the current program status for the executed steps of a GRAPH sequencer.

Requirement

- A PLC including a GRAPH instance data block has been created.
- GRAPH instance data block contains at least one tag which is visible in HMI and accessible from HMI.

  Note

  The process tag you are using for the GRAPH overview must be visible in HMI and accessible from HMI.

  To identify the tags of the GRAPH data block as visible and accessible for HMI, open the GRAPH function block, select the block in the work area, and select "Edit > Internal parameters visible/accessible from HMI" in the menu bar. Then compile the program blocks.

- An HMI device has been created.
- You have created a screen.
- The Inspector window is open.

Procedure

1. Drag-and-drop the GRAPH overview from the toolbox view into the configured screen.
2. In the Inspector window, click "Properties > Properties > Miscellaneous".
3. Open the selection button under "PLC Source > Tag".
   The "Add new object" dialog opens.
4. Select the corresponding PLC in the "Program blocks" folder.
5. Select the corresponding PLC tag of the GRAPH instance data block.

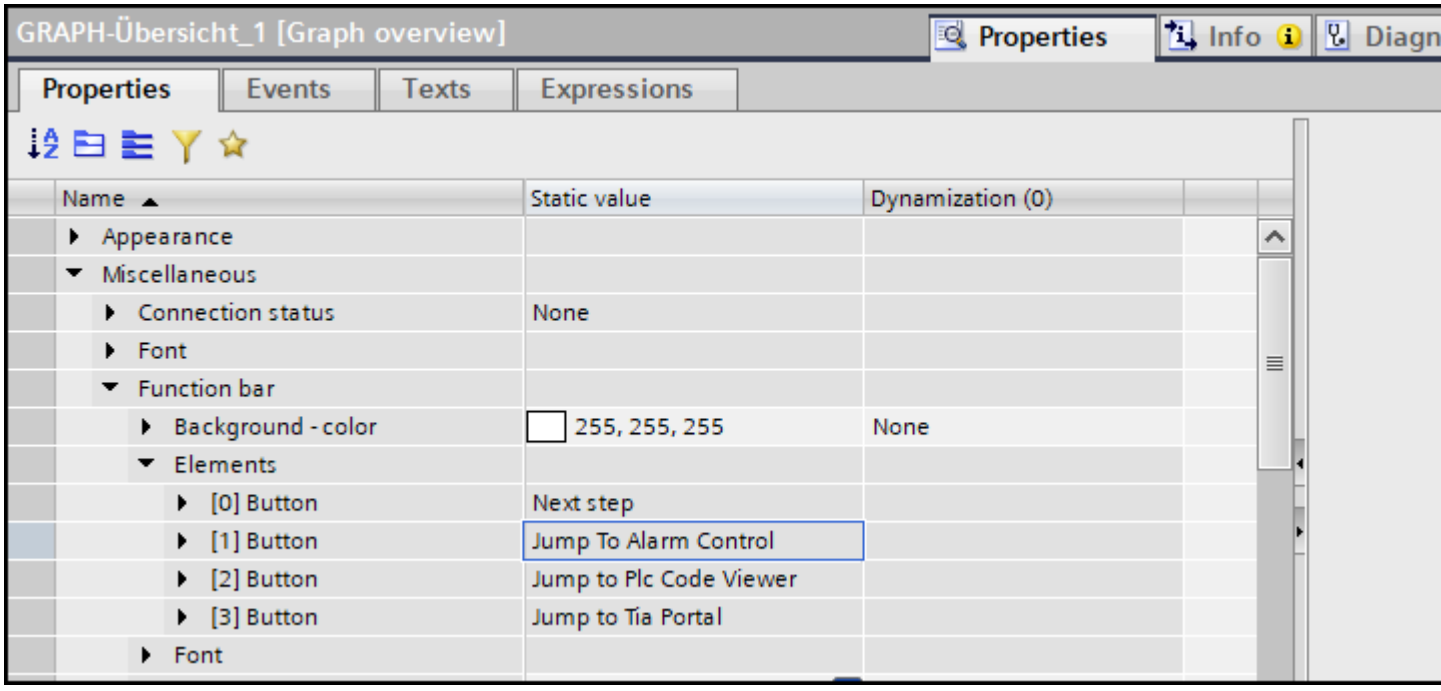
   Note

   If no connection was configured between the HMI device and the selected PLC, a connection is set up automatically.

   In addition, an HMI tag is created which is connected to the PLC tag.

6. To display the GRAPH overview in compatibility mode without toolbar buttons and operating mode display, drag the control to the desired size, whereby several basic views are possible.

7. Under "Properties > Properties > Miscellaneous > Toolbar > Elements", specify the buttons to be displayed in the object.



8. Under "Properties > Events" you can assign system functions or scripts to the buttons in the GRAPH overview in order to jump to the alarm control and the PLC code display in runtime and to open the TIA Portal.

Result

The GRAPH overview is inserted in the screen. The current status of the GRAPH step sequence is displayed in Runtime.

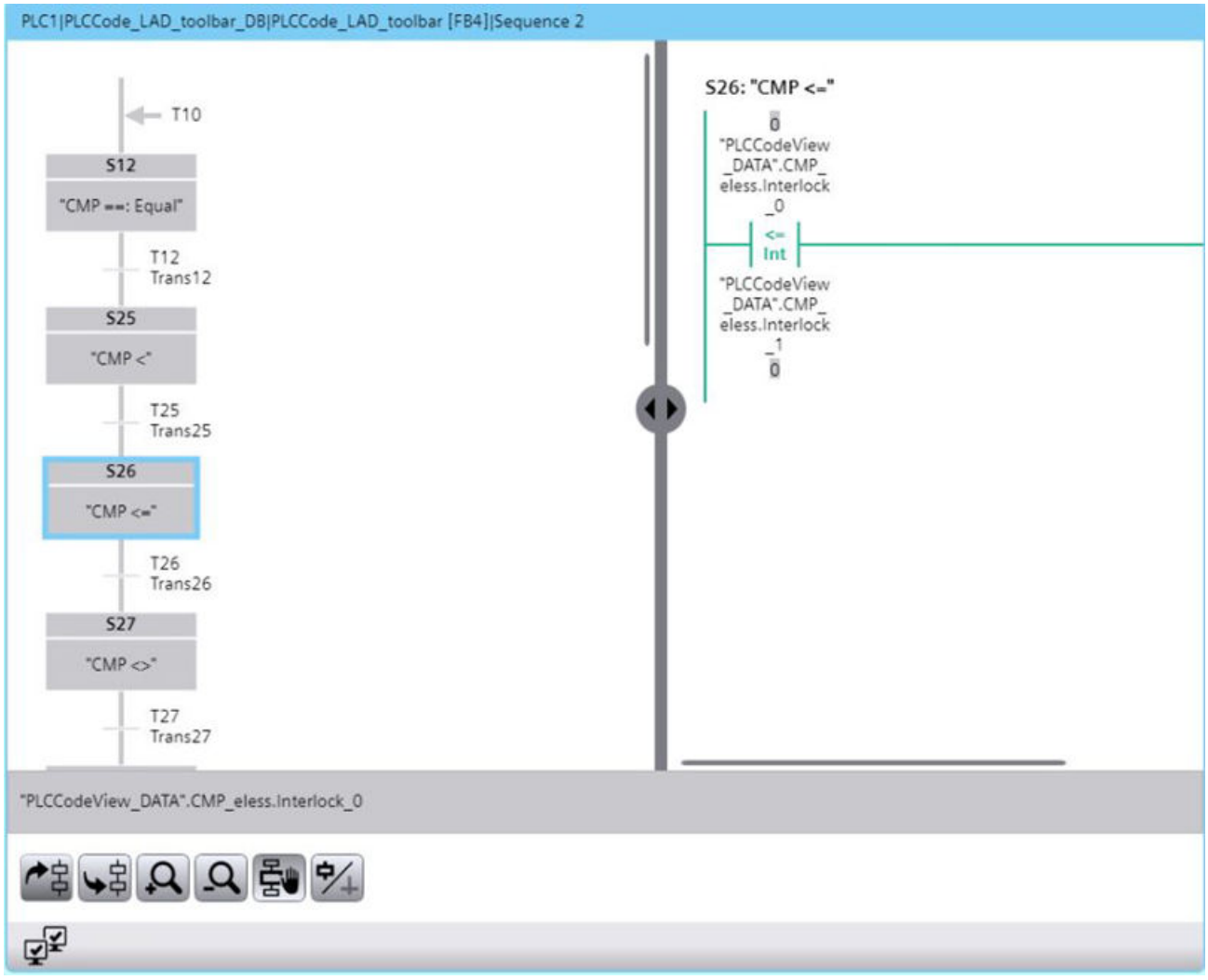
PLC code view

Use

The "PLC code viewer" object is used to display the current program status of user programs that have been programmed in the GRAPH programming language.

In the PLC code view, you display various items of information about the user program:

- Information area
- Toolbar
- Detail view
- Transition/Interlock view



Information area

The information area shows the GRAPH sequence in the left area and the details, e.g. for the step or for the transition, in the right area.

Toolbar

The toolbar shows information about the first or the selected icon.

Buttons of the toolbar

The table below shows the buttons on the toolbar and their meaning.

Operating element	Description	Function
	"Previous network"	Navigates to the previous network.
	"Next network"	Navigates to the next network.
	"Zoom in"	Enlarges the information area.
	"Zoom out"	Reduces the information area.
	"Step mode"	Switches between manual and automatic step selection for the active step.
	"Transition or Interlock"	Switches between the transition and interlock networks.

Configuring the PLC code view

Introduction

To display the PLC program networks in the GRAPH programming language in Runtime, insert a PLC code viewer control into your project.

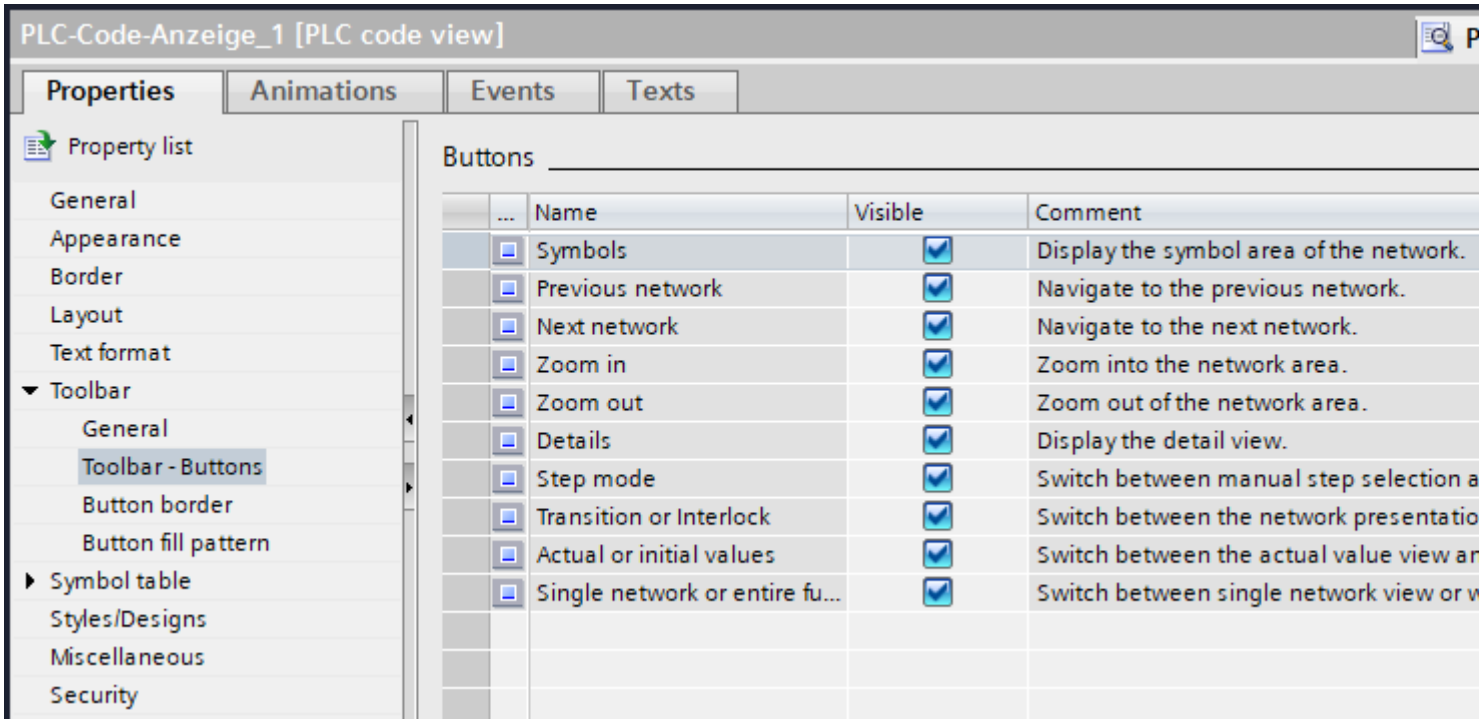
Requirement

- At least one PLC has been created.
- An HMI device has been created.
- An HMI connection has been established between the controller and HMI device.
- The system diagnostics is activated on all devices
- You have created a screen.

Procedure

1. Drag-and-drop the PLC code view from the toolbox view.
2. In the Inspector window, click "Properties > Properties > Toolbar".

3. Select the buttons that you require in Runtime, for example: Next network, Previous network, Step mode.



4. Select an authorization for operator input in "Properties > Properties > Security".

Result

The PLC code view is inserted in the screen. In Runtime, PLC user programs that are programmed in the GRAPH programming language can be displayed.

You can populate the PLC code viewer using system functions, e.g. jump from the GRAPH overview, or you can select the corresponding parameters directly.

17.3.5 Elements

17.3.5.1 Overview of elements

Operable elements are available in process pictures in Runtime.

The following elements are available depending on the configured access rights:

Icon	Element	Brief description
	Bar	Represents tags graphically. The bar graph can be labeled with a value scale.
	I/O field	Used for entry and display of process values.
	Symbolic I/O field	A drop-down list with texts or graphics for display and input in runtime.

Icon	Element	Brief description
	Check box	Used for display and selection of multiple options.
	List box	Used for display and selection of multiple list entries.
	Radio button	Used for display and selection of various options of which only one can be selected.
	Switch	Used for toggling between two predefined states.
	Button	Executes a configured function.
	Slider	Used for monitoring and changing process values within a defined range and adjusts them. By adjusting the slider, you intervene in the process and correct the displayed process value.
	Clock	Used for display of date and time.
	Gauge	Represents numerical values in the form of an analog gauge. For example, it can be seen at a glance whether the boiler pressure is in the normal range.

17.3.5.2 Using elements

Bar

Application

The tags are displayed graphically with the "Bar" object. The bar graph can be labeled with a value scale.

Layout

The settings for the position, geometry, style, colors and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Color transition: Specifies the change in color display when limit values are exceeded.
- Limit marking: Displays the configured limit value as an arrow.
- Bar segments: Defines the gradations on the bar scale.
- Scale gradation: Defines the position of the zero point on a bar scale.

If the object falls below a certain size in the light or dark style, it is automatically displayed in compact mode.

Color transition

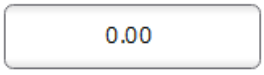
The display of the color change is specified during configuration.

Color transition	Description
"Segmented"	If a particular limit was reached, the bar changes color segment by segment. With segmented display, for example, the limits exceeded by the displayed value are visualized.
"Entire bar"	If a particular limit was reached, the entire bar changes color.

IO field

Use

The "I/O field" object is used to enter and display process values.



Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Mode: Specifies the behavior of the object in Runtime.
- Display format: Specifies the display format in the I/O field for input and output of values.
- Hidden input: Specifies whether the input value is displayed normally or encrypted during input.

Note

Reports

In reports, I/O fields only output data. "Output" mode is preset. Properties for configuring input are not available, e.g. "hidden input".

Mode

The behavior of the I/O field is specified during configuration.

Mode	Description
"Input/output"	Values can be input and output in the I/O field.
"Output"	The I/O field is used for the output of values only.

Layout

The "display format" for the input and output of values is specified during configuration.

Layout	
"Binary"	Input and output of values in binary form
"Date"	Input and output of date information. The format depends on the language setting on the HMI device.
"Date/time"	Input and output of date and time information. The format depends on the language setting on the HMI device.
"Decimal"	Input and output of values in decimal form
"Hexadecimal"	Input and output of values in hexadecimal form
"Time"	Input and output of times. The format depends on the language setting on the HMI device.
"Character string"	Input and output of character strings.

Hidden input

In Runtime the input can be displayed normally or encrypted, for example for hidden input of a password. A "*" is displayed for every character during hidden input. The data format of the value entered cannot be recognized.

Avoid overlaps in output fields

If several I/O fields are configured as output fields with a transparent background in a screen, these I/O fields may overlap. The transparent part of the one field covers the digits of the other field. This may cause display problems. In order to avoid such overlaps, the border of the I/O fields must be set to zero during configuration.

Limits

During configuration, colors can be specified for the values that exceed or fall below limits.

When there is a limit violation, the background color of the I/O field changes according to the configuration, even if the I/O field is in input mode.

A limit range can also be defined for the input in the I/O field for the configuration.

If you enter a numeric value outside this limit, it is not applied; for example, 80 with a limit of 78. In this case, a system alarm is generated on the HMI device if an alarm window is configured. The original value is displayed again.

Decimal places for numerical values

The number of decimal places can be specified for a numerical input field during configuration. The number of decimal places is checked when you enter a value in this type of I/O field. Decimal places in excess of the limit are ignored. Empty decimal places are filled with "0".

In the exponential display, the displayed numerical value is represented with a maximum precision of nine decimal places.

Setting an LTime PLC tag via HMI

S7-1500 tags with data type LTime have the unit nanoseconds (ns). IO fields that are linked with such a PLC tag have the unit 100 ns.

HMI user inputs to the I/O field are converted to ns when the value is sent to the PLC.

Note

MAX_SAFE_INTEGER

Depending on the JavaScript engine of the web client, the actual value may lose accuracy during communication between the HMI device and the controller due to rounding if it is outside the value range of MAX_SAFE_INTEGER.

Additional information on MAX_SAFE_INTEGER can be found here (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER).

Behavior when switching between input fields

When you change from one input field to another within a screen due to an operator input and the on-screen keyboard appears, the "Exit field" event is not immediately triggered for the previous field. Rather, it is only triggered after the on-screen keyboard is closed.

No events during the input

While an I/O field is in input mode, no events are transmitted to the server for the I/O field.

Terminate the input mode with Enter or Esc so that the events configured for the I/O field in engineering take effect again.

Symbolic IO field

Use

The "Symbolic I/O field" list is used for displaying texts and graphics in runtime as well as text input, if configured.

The displayed texts or graphics are assigned to the potential tag values.

Note

Selecting the default entry is not possible in runtime.

Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- "Mode": Specifies the response of the object in runtime.
- "Resource list": Specifies the text or graphic list that will be associated with the object.

Mode

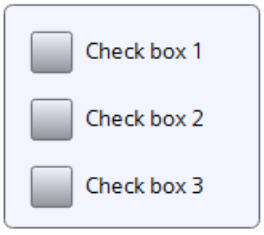
The behavior of the symbolic I/O field is specified during configuration.

Mode	Description
"Output"	The symbolic IO field is used for the output of values.
"Input/output"	The symbolic IO field is used for the input and output of values.

Check box

Application

You use the "Checkbox" object to select multiple options. Checkboxes can be activated by default so that the user changes the default values only as required. Multiple options can be selected if the corresponding properties are dynamized.



Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Number of the checkboxes: Defines the number of options.
- Selection of the checkboxes: Defines which options are displayed as activated by default.

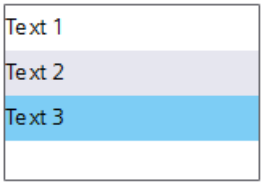
Default setting of the checkboxes

Each option is represented by a bit in a 32-bit word. To activate a field, the corresponding bit must have the value "1". The 32-bit word contains the information for all options of the checkbox list. The value of the "Presetting enabled" property is specified in hexadecimal format.

List box

Application

You use the "List box" object to present and select multiple list entries. List entries are selected by default so that the default setting can be changed only when necessary. If the list box is larger than the selection rectangle, WinCC automatically adds a scroll bar to the right margin.



Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Number of entries: Defines the number of list entries.
- Selection of entries: Defines which entry is displayed as activated by default.

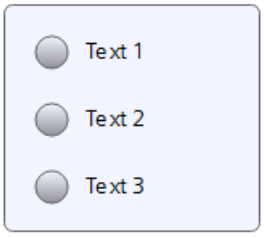
Default setting of the list boxes

Each option is represented by a bit in a 32-bit word. To activate a field, the corresponding bit must have the value "1". The 32-bit word contains the information for all texts of the list of list boxes. The value of the "Selected fields" property is given in hexadecimal notation.

Option buttons

Application

You use the "Option button" object for selection of various options. Options are selected by default so that the default setting can be changed only when necessary. Only one option can be selected if the corresponding property is dynamized.



Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Number of fields
- Selection of the fields: Specifies which fields are displayed as activated.

Switch

Application

With the "Switch" object you switch between two predefined states. The current state of the "Switch" object is visualized with either a label or a graphic.

Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following property is changed:

- Type: Defines the graphic representation of the object.

Type

The display of the switch is specified during configuration.

Type	Description
"Switch"	The two states of the "Switch" are displayed in the form of a switch. The position of the switch indicates the current state. The switch is switched by moving it.
"Switch with text"	The switch is shown as a button. The current state is visualized with a label. The switch is switched by clicking the button.
"Switch with graphic"	The switch is shown as a button. The current state is visualized with a graphic. The switch is switched by clicking the button.

Button

Use

With the "Button" object, you execute a configured function.

Layout

The settings for the position, geometry, style, color and font of the object are made during configuration.

In particular, the following properties are changed:

- Mode: Defines the graphic representation of the object.
- Text / Graphic: Defines whether the Graphic view is static or dynamic.
- Define hotkey: Defines a key, or shortcut that the operator can use to actuate the button.

  Note

  You can only define a hotkey for HMI devices with keys.

Mode

The display of the button is specified during configuration.

Mode	Description
"Invisible"	The button is not visible.
"Text"	The button is displayed with text. This text explains the function of the button.
"Graphic"	The button is displayed with a graphic. This graphic represents the function of the button.
"Graphic or text"	The button is displayed with text or graphics. If the graphic cannot be displayed, the corresponding text is displayed.
"Graphic and text"	The button is displayed with text and graphic.

Different options are available depending on the device.

Text / Graphic

Depending on the "Mode" property, the display can be specified as a static or dynamic display. The display is specified during configuration.

You can, for example, select the following options for the "Graphic" or "Text" type.

Type	Option	Description
"Graphic"	"Graphic"	With "Graphic when button "not pressed"", a graphic is specified that is displayed in the button for the "OFF" state. When "Graphic when button "pressed"" is selected, a graphic for the "ON" state can be entered.
	"Graphics list"	The graphic in the button depends on the state. The corresponding entry from the graphics list is displayed depending on the state.
"Text"	"Text"	With "Text when button "not pressed"", a text is specified that is displayed in the button for the "OFF" state. When "Text when button "pressed"" is selected, a text for the "ON" state can be entered.
	"Text list"	The text in the button depends on the state. The entry from the text list corresponding to the state is displayed.

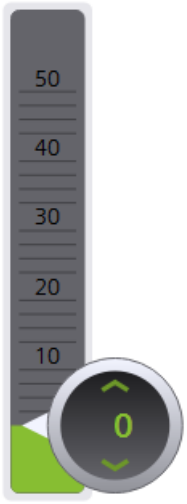
Hotkey

A key or key combination that the operator can use to actuate the button can be defined during configuration.

Slider

Use

Process values are monitored and adapted within a defined range with the "Slider" object. The monitored range is visualized in the form of a slider. By adjusting the slider, you intervene in the process and correct the displayed process value.



Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Maximum Value and Minimum Value: Specifies the top and bottom values of the scale.
- Display current value: Specifies whether the current position of the controller appears below the slider.
- Display of bars: The sliders above and below the bar can be hidden.

If the object falls below a certain size in the light or dark style, it is automatically displayed in compact mode.

Limits/ranges

You can represent limits and ranges in different colors. The colors are defined during configuration.

Note

If the "Show ranges from tag" option is selected, up to five ranges can be displayed in a slider. The values of the ranges are specified using a process tag. The values for the ranges are defined with a process tag that is connected to the screen object.

The option "Show ranges from tag" is available for Comfort Panels, KTP Mobile Panels and RT Advanced.

Behavior during operation

The displayed value on the slider control may deviate from the actual value of the associated tag in the following circumstances:

- The value range (minimum and maximum value) configured for the slider control does not correspond to the configured limits for the slider control tag.
- An invalid password has been entered for a password-protected slider control.

Clock

Application

The "Clock" object displays the date and time.

By default, the "Clock" object displays the date and time of the client.

If the "Process value" property of the clock is connected to a DateTime tag, the clock uses the tag value as a start value and continues counting. When the tag value is changed the time is synchronized and continues counting from the new value.

Note

Static display of a date-time value

If the image is supposed to display a static time of day, link a tag of the type DateTime with an I/O field.

Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are adapted:

- Analog display: Specifies whether the clock is shown as an analog clock or digital clock.
- Display clock dial: Specifies whether hour marks of the analog clock will be displayed.
- Width and length of hands: Specifies the width and length of the hands.

If the object falls below a certain size in the light or dark style, it is automatically displayed in compact mode.

Gauge

Use

The "Gauge" object shows numeric values in the form of an analog gauge. For example, it can be seen at a glance whether the boiler pressure is in the normal range.

Note

The gauge is for display only and cannot be controlled by the operator.

Layout

The settings for the position, geometry, style, color and fonts of the object are made during configuration.

In particular, the following properties are changed:

- Display peak value: Specifies whether the actual measuring range is indicated with a peak indicator.
- Maximum Value and Minimum Value: Specifies the top and bottom values of the scale.
- Start value of the danger range and start value of the warning range: Specifies the scale value from which the danger range and the warning range start.
- Display normal range: Specifies whether the normal range is shown in color on the scale.
- Color of individual ranges: Different operating modes, such as normal range, warning range and danger range, are shown in different colors so that the operator can distinguish them easily.

If the object falls below a certain size in the light or dark style, it is automatically displayed in compact mode.

Note

The use of many differently sized "Gauge" objects can reduce the performance in Runtime. With "Gauge", avoid minimally different heights and widths, for example, 48 pixels, 49 pixels, 51 pixels, etc. Use the same sizes instead.

Display peak value

The "Display peak value" property can be used to enable a marker function for the maximum and minimum pointer movement in Runtime. The actual measuring range is shown with a min/max pointer.

Color of individual ranges

The normal range, danger range and warning range can be displayed in different colors. The colors are defined during configuration.

Note

If the "Show ranges from tag" option is selected, up to five ranges can be displayed in the gauge. The values of the ranges are specified using a process tag. The values for the ranges are defined with a process tag that is connected to the screen object.

The option "Show ranges from tag" is available for Comfort Panels, KTP Mobile Panels and RT Advanced.

17.3.6 Basic objects

In addition to controls and elements, HMI screens contain basic objects such as circles, polygons or text boxes. Basic objects are often used for design purposes, but can also provide information about the process.

Dynamically configured basic objects react to changes in the process or to operator actions. Example: In engineering, a text box is linked to a text list that defines text entries for the value range of a tag. In Runtime, the text box always shows the text assigned to the current tag value. When the tag changes its value, the content of the text box changes.

Overview of basic objects

Depending on the configuration, screens can contain the following basic objects:

- Line
- Polyline
- Polygon
- Ellipse
- Ellipse segment
- Circle segment
- Elliptical arc
- Circular arc
- Circle
- Rectangle
- Text box
- Graphic view

Process values in text fields

If a text box has been connected to a tag in engineering, the text box shows the process value of the tag in runtime.

If the text box was connected to a tag and a text list, the text box shows the text list entry that corresponds to the tag value.

Note

If no default value is assigned to the text list and the tag value is outside the defined value range of the text list, the last valid process value displayed by the text box is output.

17.3.7 Popup window

Popup windows are freely movable windows that open when an event configured in the engineering system occurs. They show, for example, additional information on a partial area of the process image.

You close a popup window using the button in the top right corner of the popup window.

Example

Runtime shows a screen with an overview graphic for a pump and its valves.

Configuration in the engineering system

A faceplate instance was positioned on the screen for each valve, which is displayed by the graphic of the valve. The faceplate instances have a script that opens an additional faceplate instance in a popup window in Runtime. This second instance shows detailed information on the valve as well as input fields.

Behavior in Runtime

When you click on a valve in the overview graphic in the screen, a popup window opens. In the popup window, you can check the state of the valve and edit the valve using the input fields.

17.3.8 Tests and error analysis

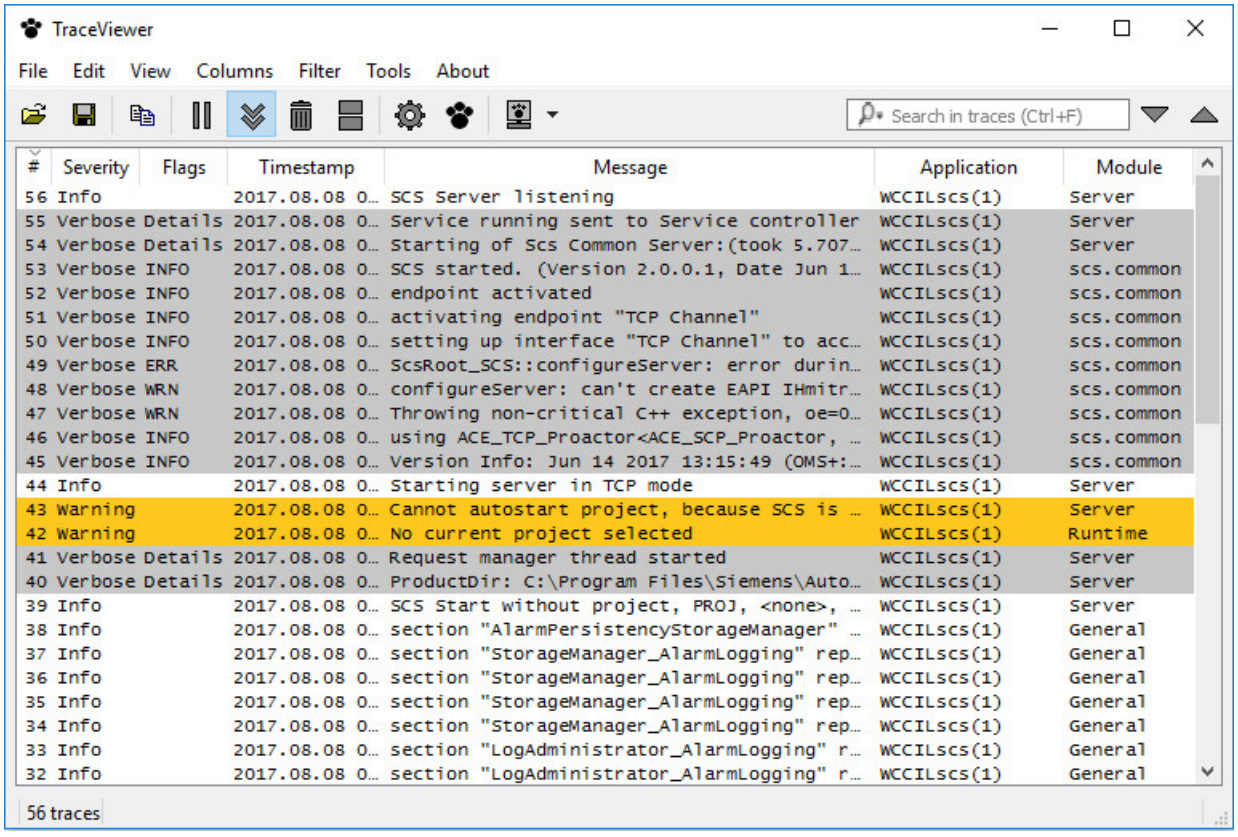
17.3.8.1 Trace logs for function calls and tag values

WinCC Unified provides trace logging for error analysis. Tag values and function calls can be logged for test purposes and for troubleshooting with the trace.

All trace outputs with "Fatal", "Error" or "Warning" severity are stored in LOG files (.log) in the directory "%ProgramData%\Siemens\Automation\Logfiles\WinCC_Unified_SCADA_Vxx". In case of problems you must send these files to SIEMENS Customer Support.

TraceViewer

The LOG files can be viewed with the TraceViewer. It is located in the installation directory of WinCC Unified under "WinCCUnified\bin". To open the Trace Viewer start the file "RTILtraceViewer.exe".



17.3.8.2 Debugging scripts

Basics of debugging

Introduction

For example, you can use a debugger to test whether correct values are being transferred to tags and whether abort conditions are being correctly implemented. Check the following in the debugger:

- Source code of functions
- Function sequence
- Values

Note

Your code is displayed in the debugger but is write-protected.

Basic procedure

To find an error, check the script with the debugger.

The following options are available for your support:

- Setting breakpoints
- Step-by-step execution
- Viewing values parallel to execution of the script

You do not edit the code of your scripts directly in the debugger. When you find an error, follow these steps:

1. Correct the error in the engineering system.
2. Compile the changed code.
3. Load the runtime.
4. Update the debugger.

Design and function of the debugger

Google Chrome provides the user interface of the debugger. Not all functions of the user interface of the debugger are relevant for debugging WinCC Unified Scripts. Only the functions that are needed to debug scripts in WinCC Unified are explained below.

You can find more information on Chrome DevTools under: https://developers.google.com/web/tools/chrome-devtools/.

The debugger is divided into two areas:

- Debugger for screens
- Debugger for jobs

With the debugger for screens you view scripts at screens and screen objects. With the debugger for jobs, you view scripts that you have configured in the Scheduler.

Start page of the debugger

After the debugger has been started, its start page is displayed.

The available contents differ depending on the selected area.

On the start page of the debugger for screens you can see two different contexts:

- Dynamizations (e.g. "UMCadmin@192.168.116.144 VCS_1 Dynamics")
- Events (e.g. "UMCadmin@192.168.116.144 VCS_1 Events")

The name of the contexts is composed as follows:

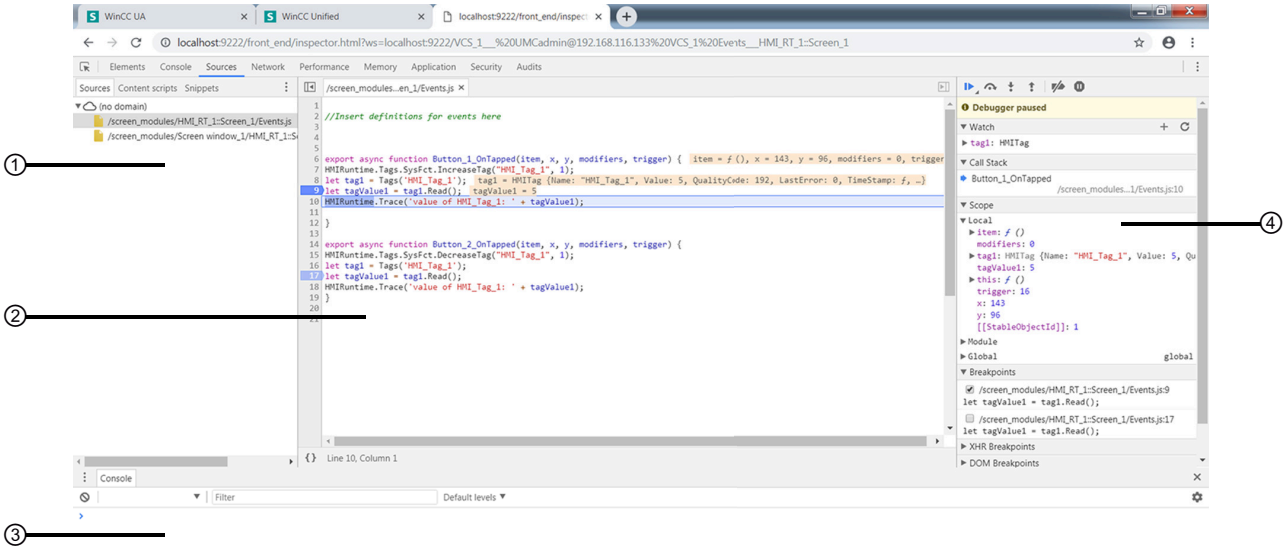
- UMCadmin: User name
- 192.168.116.144: IP address of the computer
- VCS: Name of the graphic component
- _1: Number of the open client
- Events/Dynamics: Scripts at events or dynamizations

Note

A client corresponds to a tab in Google Chrome in which the runtime is open. When you have opened runtime in multiple tabs, multiple clients are used. The client opened first is given the number 1. Numbering is reset when the runtime is restarted.

On the start page of the debugger for jobs you can see the context "JobsExecution".

User interface of the debugger



① Navigation area
② Code display area
③ Console
④ Debugging area

Navigation area

In the navigation area, the available contents for the screen shown in runtime are displayed in groups. The available groups vary depending on the use of scripts and functions.

Groups in the debugger for screens

The debugger for screens can contain the following groups in the dynamizations context:

- A group for scripts that were configured for dynamizations.
- One group per screen window in which scripts were configured for dynamizations.

The debugger for screens can contain the following groups in the events context:

- A group for scripts that were configured for events.
- One group for functions that were configured for events using the function list.
- One group per screen window in which scripts were configured for events.
- One group per screen window in which functions were configured for events using the function list.

Groups in the debugger for jobs

The debugger for jobs can contain the following groups:

- A group for scripts that were configured for tasks.
- One group for functions that were configured for tasks using the function list.

Code display area

Your code is displayed in the code display area. The rows are numbered.

Debugging area

The debugging area offers the following relevant options for WinCC Unified:

- Toolbar: Control for executing the script
- "Watch": Display of values
- "Callstack": Display of the current call stack
- "Scope": Available local values ("Local"), functions ("Module") and global values ("Global"),
- "Breakpoints": List of set breakpoints

Enabling the debugger

Requirement

- SIMATIC Runtime Manager is installed.
- The logged-on user belongs to the Windows user group "SIMATIC HMI".

Note

The debugger is only available locally.

Remote access from the debugger to other devices is not possible.

Procedure

The debugger is disabled by default.

Note

The debugger should be disabled in production operation, as using the debugger can endanger system stability and security. Actions can accumulate if the debugger is, for example, at a breakpoint for a long time or the screen is not refreshed.

To enable the debugger, follow these steps:

1. Start the SIMATIC Runtime Manager application.
2. Click the  button in the toolbar.
3. Switch to the "Scripts Debugger" tab.
4. To enable the debugger for screens, select the "Enable" check box in the "Screen debugger" area.
5. To enable the debugger for scheduled tasks, select the "Enable" check box in the "Scheduler debugger" area.
6. Assign an available port number to the debugger for screens (default port number: 9222).
7. Assign an available port number to the debugger for jobs (default port number: 9224).
8. Confirm your entries.

Note

Start Runtime after enabling the debugger.

Starting the debugger

Requirement

- Google Chrome (as of version 70) is installed.
- A project is opened in runtime.
- The debugger was activated in SIMATIC Runtime Manager.

Note

The debugger is only available locally.

Remote access from the debugger to other devices is not possible.

Procedure

1. In a new tab, call up the URL chrome://inspect in Google Chrome.
2. The home page of the Chrome DevTools is loaded in the tab.
3. Click "Devices".
4. Select the "Discover network targets" check box.
5. Click "Configure".
6. In the "Target discovery settings" dialog box, enter one of the following strings:
   - `127.0.0.1:<Port number>`
   - `localhost:<Port number>`

   Use the port entered for the Script Debugger in SIMATIC Runtime Manager.
7. Press <Enter>.
8. Click "Done".
9. Under "Remote Target", click "inspect" for the desired target.
   The DevTools open in a separate window with the selected target.
10. In the DevTools, select "Sources".
    The debugger is displayed.
11. Click "Toggle screencast".
12. In the navigation area under "Page", select the desired script module.

Updating the debugger

The debugger must be updated:

- After starting a new project
- After restarting a running project, for example, because you have reloaded the project in engineering with "Download to device > Software (all)".
- After a screen change in Runtime

The connection to the debugger is lost in each case. Google Chrome therefore shows an error message and asks whether you want to restore the connection.

To restore the connection, proceed as follows:

1. Close the DevTools window.
2. On the DevTools start page under "Remote Target", click "inspect" again for the desired target.

Stopping the debugger

Exit the debugger by closing the DevTools window and, if necessary, the DevTools homepage.

This does not stop runtime.

Working with breakpoints

Set breakpoints to stop the execution of the script at certain points and thus localize errors step-by-step. Previously set breakpoints are still available after updating the debugger.

Requirement

- Runtime has started.
- The debugger has been started.
- The group you want to debug is selected.

Pause script

To pause the execution of a script, you have 2 options:

- To pause the script immediately, click the "Pause script execution" button while the script is being executed.
- Set a breakpoint in the desired line.
  The script pauses when a breakpoint is reached.

To pause a script at a breakpoint that is configured to an event, follow these steps:

1. Set a breakpoint in the script.
2. Trip the respective event in runtime.
   The script pauses at the breakpoint.

Setting breakpoints

You have several options to set a breakpoint in a line of the script:

- Click on the line number.
- Right-click the line number and select "Add Breakpoint".

All set breakpoints are displayed in the debugging area under "Breakpoints".

Linking breakpoints to conditions

To link a breakpoint to a condition, proceed as follows:

1. Open the shortcut menu of the relevant line.
2. Select the entry "Add conditional breakpoint".
   Execution of the script is stopped at the breakpoint when the condition is fulfilled.

Edit conditions as follows:

1. Open the shortcut menu of the relevant line.
2. Select the entry "Edit breakpoint...".

To prevent the script from pausing at a selected line, proceed as follows:

1. Open the shortcut menu of the respective line.
2. Select the entry "Never pause here".

Showing and hiding breakpoints

When you hide a breakpoint, its position is retained. The script then ignores the hidden breakpoint. When you need the breakpoint again, it can simply be shown.

In the debugging area, all breakpoints set in the selected group are displayed under "Breakpoints".

You have several options to show a breakpoint:

- Set the check mark in front of the relevant breakpoint in the debugging area under "Breakpoints".
- Alternatively, right-click the number of the respective line in the code display area and then select "Enable breakpoint".

You have several options to hide a breakpoint:

- Remove the check mark in front of the relevant breakpoint in debugging area under "Breakpoints".
- Alternatively, right-click the number of the respective line in the code display area and then select "Disable breakpoint".

To show or hide all breakpoints, follow these steps:

1. Open the shortcut menu in the debugging area under "Breakpoints".
2. Select "Enable all breakpoints" or "Disable all breakpoints"

Enabling and disabling breakpoints

You can enable or disable all breakpoints independent of showing or hiding individual breakpoints.

You have several options to enable or disable all breakpoints:

- Click on the  "Deactivate breaktpoints" button in the debugging area.
- Open the shortcut menu of a breakpoint in the debugging area and select "Activate breakpoints" or "Deactivate breakpoints".
- Press <Ctrl + F8>.

Deleting breakpoints

You have several options to delete a breakpoint:

- Click on the breakpoint in the code display area.
- Open the shortcut menu of the breakpoint in the code display area and select "Remove breakpoint".
- Open the shortcut menu in the debugging area under "Breakpoints" and select "Remove breakpoint"..

To delete breakpoints, the shortcut menu offers the following additional options in the debugging area under "Breakpoints":

- Delete all breakpoints ("Remove all breakpoints")
- Delete all breakpoints except the selected breakpoint ("Remove other breakpoints")

Step-by-step execution of scripts

Introduction

The following options are available to execute your script step-by-step:

- Execute script to the next breakpoint
- Force execution of a script
- Execute script to the next function call
- Jump into a function
- Jump out of a function
- Execute script up to a selected line
- Pause at Exceptions
- Use call stack

Requirement

- The group you want to debug is selected.
- The script pauses at a breakpoint.

Execute script to the next breakpoint

To pause the continuation of a script, you have several options:

- Click on the "Resume script execution" button in the debugging area.
- Press the <F8> key.
  The script is executed to the next breakpoint. If there is no other breakpoint, the script is executed completely.

Force execution of a script

To ignore the following breakpoints when resuming execution of a paused script, proceed as follows:

1. Click and hold down the "Resume script execution" button.
   The "Force script execution" button appears.
2. Move the mouse pointer to the "Force script execution" button while keeping the mouse button pressed.
3. Now release the mouse button.
   The script is executed to the end.

Execute script to the next function call

If a line with a breakpoint contains a function that you are not interested in, you can suppress the debugging of this function:

- Click on the "Step over next function call" button in the debugging area.
- Press the <F10> key.
  The function is executed without the script pausing within the function.

Jumping into a function

If the script pauses in a line containing a function that interests you, you can pause the script in that function:

- Click on the "Step into next function call" button in the debugging area.
- Press the <F11> key.
  The script pauses in the first line of the function.

Note

You can only jump into functions that you have defined yourself.

Jump out of a function

If the script pauses within a function that you are not interested in, you can suppress further debugging of this function:

- Click on the "Step out of current function" button in the debugging area.
- Press the key combination <Shift + F11>.

Note

You can only jump out of a function that you have defined yourself.

Execute script up to a selected line

To pause a paused script again at a selected line, proceeds as follows:

1. Right-click the number of the line in the code display area.
2. Select the entry "Continue to here".
   The script pauses at the selected line.

Pause at Exceptions

- To pause the script at Exceptions, click on the "Pause on exceptions" button in the debugging area.

Use call stack

- To jump into a function of the call stack, click on the corresponding entry under "Call Stack".

Note

You can only jump into functions that you have defined yourself.

Show values

Introduction

To identify errors in your script efficiently, have current values displayed while the script is being executed. This way you can view properties of objects or parameters of functions, for example. You can find additional information on objects and their properties under "WinCC Unified Object Model".

Requirements

- The group you want to debug is selected.
- The script pauses at a breakpoint.

Procedure

You view values by moving the mouse over the label in the code display area.

You also have the following options to view values:

- In the debugging area under "Scope"
- In the debugging area under "Watch"
- In the console

"Scope" area

All local values ("Local"), functions ("Module") and global values ("Global") that are defined at this time are displayed in the "Scope" area.

The values cannot be edited.

"Watch" area

In the "Watch" area, you view how values change in the course of a script.

The following buttons are available to you:

- "Add expression": Add a value
- "Refresh": Refresh the "Watch" area
- "Delete watch expression": Delete a value from the "Watch" area. Available when the mouse pointer is located above the respective value.

Console

The values available at the current time can be called in the console.

- You show or hide the console with <Esc>.

Call the current values in the console as follows:

1. Enter the name of a local or global value in the console.
2. Press <Enter>.

17.4 SIMATIC Runtime Manager

17.4.1 Functions in the SIMATIC Runtime Manager

Introduction

The SIMATIC Runtime Manager offers the following options for WinCC Unified PC:

- Use the project list to get an overview of the projects loaded into the Runtime and their properties.
  See The Runtime Manager user interface (Page 7493).
- Manually start and stop a project loaded into the Runtime.
  See Starting the project (Page 7495).
- Define a project that is started automatically when the HMI device starts up.
  See Selecting an autostart project (Page 7499).
- Restore log segments in Runtime and delete restored segments.
  See Restoring and deleting log segments (Page 7499).

- Load a project from an external storage medium into Runtime.
  See Adding a project (Page 7497).
- Make the following settings, if required:

Password	Enter the password that is used by the Runtime Manager for secure communication with Runtime.	See Enter password (Page 7500).
Autoscaling	Enable automatic adaptation of the HMI screens to the window size of the browser in which the Runtime project is displayed (autoscale).	See Setting general settings (Page 7501).
Language	Change the user interface language of SIMATIC Runtime Manager.	
Automatic login	Enable automatic login for a local web client of a Unified PC.	See Activating automatic login (Page 7502).
Start external processes	Enable the start of external processes from runtime.	See Allowing start of external processes (Page 7503).
OPC UA Export	Export the tags of the project running in Runtime into an XML file via the OPC UA server.	See Exporting tags via the OPC UA server (Page 7507).
User management	Enable the user administration of the project running in Runtime.	See Activating user management (Page 7508).
Certificates	Manage and distribute certificates of external communication partners and manage and distribute the root certificate of the Unified PC.	See Managing certificates (Page 7504).
Runtime script debugger	Configure and enable the Runtime script debugger (screen debugger and scheduler debugger).	See Setting the Runtime Script Debugger settings (Page 7509).

17.4.2 Start Runtime Manager

Requirement

WinCC Unified Runtime for PC is installed on the device.

Procedure

Double-click the desktop link created during the installation of WinCC Unified Runtime.

Alternatively, start the Runtime Manager from a file explorer by double-clicking the following file: "<Path to the Unified installation directory>\bin\SIMATICRuntimeManager.exe"

For example C:\Program Files\Siemens\Automation\WinCCUnified\bin\SIMATICRuntimeManager.exe

Note

Starting the Runtime Manager as administrator

Some settings under "Settings" require the Runtime Manager to be started as administrator. Right-click on the .exe and select "Run as administrator".

17.4.3 The Runtime Manager user interface

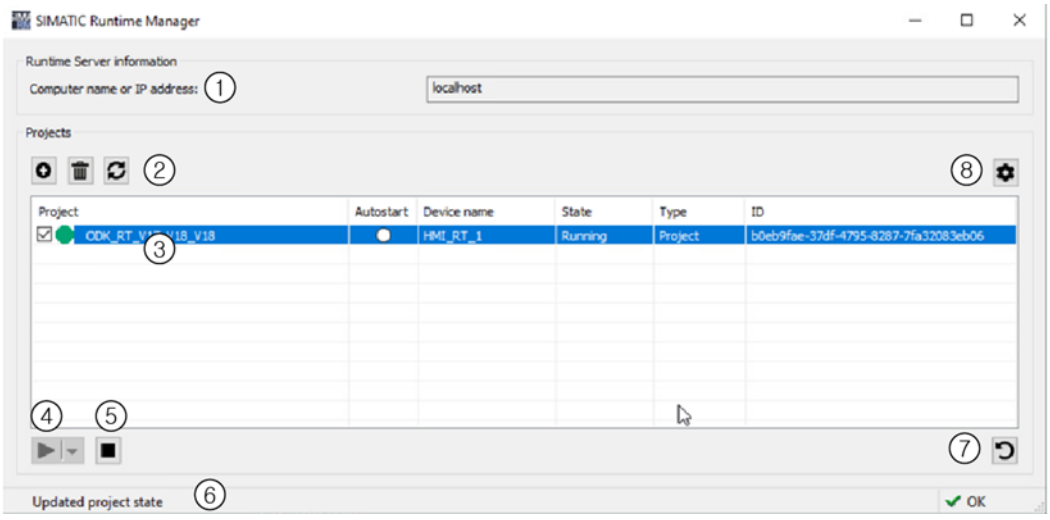
Note

User interface language

Runtime Manager starts with the language configured in the general settings. You can change the interface language. See also Setting general settings (Page 7501).

Structure

The Runtime Manager has the following structure:



① Information about the server on which the Runtime is installed
② Toolbar
③ Project list
④ Button to start the project is selected in the project list
⑤ Button to stop the project is selected in the project list
⑥ Information bar
⑦ "Restore/remove database segments for logs" button
⑧ "SIMATIC Runtime Manager settings" button

Toolbar

The toolbar has the following buttons:

Icon	Function
	Loads a project from an external storage medium into the Runtime.
	Deletes from the Runtime the project selected in the project list. The project folder and the log folders are deleted.
	Updates the project list.

Content of the project list.

The project list shows all projects loaded into the Runtime.

The list provides the following information on the projects:

Project details	Description
Project	Project name
Autostart	Indicates whether the "Autostart" option is enabled.
Device name	Device name
State	State of the associated Runtime service Possible status values: • Running • Partly running • Shutting down • Stopped • Unknown
Type	Type of the Runtime service Project: Runtime mode Simulation: Simulation mode
ID	Project-ID

17.4.4 Starting the project

Requirement

A project is loaded in runtime that does not have the "Running" state.

Start without reset

Proceed as follows to start the project in a state that existed before the last project stop:

1. Click on the project in the project list.
2. Click the "Start" button ▶|▾.
3. Select "Start".

Start with reset

Proceed as follows to start the project in a state that existed during the first project start:

1. Click on the project in the project list.
2. Click the "Start" button ▶|▾.
3. Select "Start with options".
4. Enable the options "Reset logging data" and/or "Reset Runtime data" in the "Start project options" dialog.
5. Click "Start".

"Partly running" status

If it is not possible to start the simulation or the Unified Runtime and the status of the project is displayed as "partly running", check the following:

- Does the user currently logged on in Runtime have sufficient rights? Is the user registered in the following Windows user groups:
  - PlcSimUsers
  - RTIL Tracing Users
  - Siemens TIA Engineer
  - SIMATIC HMI
  - SIMATIC HMI VIEWER
- Is the computer name no longer than 15 characters?
- Is the "OPC UA" activated in the Runtime settings and is a certificate is available?
- Is "Runtime Collaboration" enabled in the Runtime settings of a Unified Panel and is a certificate available?

Result

- The project is started.

  Note

  Activating user management

  The login to the Runtime project requires that its user management is active in Runtime.

  After starting a project manually, you have to activate its user management manually.

- If the "Reset logging data" option was enabled, the following data is deleted when Runtime is started:
  - Logging tags
  - Log alarms
  - Logged context values
- If the "Reset Runtime data" option was enabled, the following data originating from the last runtime of the project is deleted when Runtime is started:
  - The last values of internal, persistent tags
  - The last alarm states
  - The persistent attributes of the alarm system
  - The persistent attributes for the last logging cycle of the logging tags.

See also

Activating user management (Page 7508)

17.4.5 Adding a project

You have the option of loading projects from an external storage medium into Runtime with the Runtime Manager.

Requirement

- The external storage medium with the Runtime project is connected to the computer.
- The Runtime Manager is open.
- To download a project for which only the changes to the project have been downloaded to the external storage medium, the following additional requirements must be met:
  - The project that is to receive the changes is running on the HMI device.
  - The Runtime ID of the running project and the project on the external storage medium match.

Procedure

1. In the toolbar, click "Add project from offline transmission": 
   The "Add projects" dialog box opens.
2. Under "Select project log", click "...".
   A selection dialog opens.
3. Select the compressed ZIP folder of the Runtime project on the storage medium.
4. Click "Open".
   Under "Project information" you can see details of the selected project.

5. For a project that has been completely loaded onto the external storage medium, set the following options:
   - To start the project in Runtime after loading, select the "Start Runtime with project" option under "Options".
     Alternatively, you can start the project later in Runtime Manager.
   - Define whether project data is reset on startup.
     To start the project in a state that existed when the project was first started, activate the options "Reset logging data" or "Reset Runtime data".
     Disable these options to start the project in a state that existed before the last project stop.
     For more information on which data is reset with these options, see section Starting the project (Page 7495).
   - Under "Check IDs", determine whether or not to check the synchronization of IDs between engineering data and runtime data.
     Check activated: If inconsistent IDs are reported, the download is canceled. The IDs are then not synchronized.
     Check not activated: The project is loaded without checking. The system cannot guarantee that the data loaded from the Engineering System match the data present in Runtime.

   Note

   Restart Runtime

   To prevent data inconsistencies, restart Runtime when you select "Do not Sync".

6. To overwrite the Runtime UMC data with UMC data from the project, select the "Overwrite UMC data with the context of the offline loading" option under "Options".
7. Confirm with "Add project".

Result

- The project is downloaded to Runtime and appears in the project list.
- When "Start Runtime with project" is activated: The running project is stopped and the downloaded project is started. Depending on your settings, the Runtime data and log data of the project is reset and the Runtime UMC data is overwritten by the UMC data from the project.

Note

When you load a project from an external storage medium, the SIMATIC Runtime Manager extracts the repository to a temporary folder on the target system. The transfer to Runtime takes place from this folder, which is then deleted again.

17.4.6 Selecting an autostart project

Requirements

- At least one project is loaded into Runtime.
- The Runtime Manager is open.

Procedure

In the project list for the desired project, select the option in the "Autostart" column.

Note

Restrictions

- You can only select one project for autostart at a time.
- The project must not have the "Simulation" project type.

Result

The project is started automatically when the device on which the Runtime is installed is started.

17.4.7 Restoring and deleting log segments

In Runtime you have the option of restoring segments from logs for which a backup was created.

You can visualize the restored data in a trend control, for example.

Note

Database type for backups

Backups can only be created if the Microsoft SQL database type is used.

You can find more information on logs in the help of the TIA Portal.

Requirement

- At least one backup of a tag or alarm log is available.
- A project is loaded into Runtime and is in the "Running" state.
- The Runtime Manager is open.

Restoring log segments

1. Select the project.
2. Click "Restore/delete database segments for logs".
   A dialog opens.

3. Select the log type:
   - "Alarm" for alarm logs
   - "Tag" for data log
4. If required, select the relevant log in the selection menu.
5. If required, define a start time or end time.
   If you define a start time, all entries from this point in time are restored.
   If you define an end time, all entries up to this point in time are restored.
   If you define a start time and an end time, all entries between the defined points in time are restored.
6. If you have moved the backup of the log to be restored, enter the changed storage path of the backup under "Backup path".

   Note

   Only one log can be restored using the "Backup path" option.

7. Click "Restore segments".
   The selected segments are restored.
   If you have selected a time period, data may be restored beyond the selected period, as the restoration is carried out segment by segment.
   Information on the restoration can be found under "Status".

Delete log segments

To delete all previously restored segments of the tag logs or alarm logs, follow these steps:

1. Select the log type:
   - "Alarm" for alarm logs
   - "Tag" for data log
2. Click "Delete segments".

   Note

   All restored segments of the selected log type are deleted regardless of the log or the defined period.

   Information on the deletion process can be found under "Status".

17.4.8 Enter password

For secure communication with Runtime, the same password must be stored in the Runtime Manager as in Runtime.

Requirement

The Runtime uses secure communication.

Note

Enabling secure communication

Secure communication for Runtime can be enabled as follows:

- During the installation of the Runtime, in the step "Secure Download";
  Or after the installation in the application "WinCC Unified Configuration".
- In the Engineering System, if encrypted transmission is configured in the Runtime settings of a device and the option "Allow initial password transfer via unencrypted download" is enabled when downloading the device to Runtime.
  After the first, unencrypted transmission, the Runtime switches to secure communication.

Enter password for secure communication

1. Click the button in the toolbar.
2. Select the "General" tab.
3. Under "Secure connection", enter the same password that is used by Runtime for secure communication.
   For more information, refer to the "SIMATIC Unified PC Installation" user help under "Secure download".

Note

If Runtime does not use secure communication, the password entered here is ignored during communication with Runtime.

17.4.9 Setting general settings

Enable Autoscale

Proceed as follows to automatically adapt the size of HMI screens to the window size of the browser in which a Runtime project is open:

1. Click the button in the toolbar.
2. Select the "General" tab.
3. Under "Autoscale", select the "Adapt screen to window" check box.
4. Restart the currently running project or start another project that is loaded into the Runtime.

When users zoom in or out of the browser window, the HMI screens automatically adapt. Users always see the entire screen.

Changing the user interface language

Proceed as follows:

1. Click the button in the toolbar.
2. Select the "General" tab.
3. Select a language under "Language > Select language".
4. Click "OK".
   Changing the interface language requires you to restart the Runtime Manager. To restart the Runtime Manager directly, confirm the message that opens with "OK".

17.4.10 Activating automatic login

Introduction

Automatic login to Runtime can be enabled for local web clients of a Unified PC.

A local web client is a web client located on the same HMI device as Unified Runtime.

Requirement

- The HMI device is a Unified PC.

Procedure

1. On the HMI device, open the SIMATIC Runtime Manager as administrator.
2. Click the button in the toolbar.
3. Select the "General" tab.
4. Under "Automatic login", activate the "Enable automatic login" option.
5. Enter the user name and password of the UMC user that automatic login is to use if no UMC user is logged into Runtime via UMC Desktop Single Sign-on yet when the local web client is started or on login to Runtime.
6. Confirm your entries with "OK".
7. Restart Runtime.

Result

- On the start of a local web client or on connection to Runtime, the web client automatically authenticates itself with the following user data:

A UMC user is already logged in on the HMI device via UMC Desktop Single Sign-on (DSSO)	
Yes	The logged-in user is used.
No	The user configured in SIMATIC Runtime Manager is used. If no user has been configured in SIMATIC Runtime Manager, a hard-coded default user without function rights is used.

  All local web clients use the same logged-in user.
- Operators see the start screen of the project running in Runtime.
- If the logged-in user does not have the authorization to operate a screen element, a login dialog opens.
  To operate the screen element, the operator must log in with a user with corresponding function rights. The open process screens remain open.
- After logout of the user used for automatic login, for example, via the "LogOff" system function, or after switchover to another user, automatic login is only possible again after Runtime is restarted.
  Logout takes effect in all applications that use DSSO. The local web client switches to the hard-coded default user without function rights. The open process screens remain open.

17.4.11 Allowing start of external processes

Introduction

System functions that start an external process in runtime can be configured in the engineering.

Example: The `OpenTIAPortalProject` system function, which starts the TIA Portal, can be called in runtime in the process diagnostics.

Procedure

To allow runtime to start an external process when such a system function is called, follow these steps:

1. On the HMI device, open the SIMATIC Runtime Manager.
2. Click the  button in the toolbar.
3. Select the "Security" tab.
4. Enable the "Allow start of external processes via Unified Runtime" option.

17.4.12 Managing certificates

External communication partners are devices that exchange data with WinCC Unified Runtime via a secured connection. These devices use certificates for authentication that are either self-signed or issued by another certificate authority.

Examples: An S7PLUS device or an external OPC UA communication partner

Introduction

The "Certificates" tab provides you with the following options:

- Manage the trusted certificates of the external communication partners of the Unified HMI device (certificates, root certificates, and CRL files).
- Export the certificates of the external communication partners to distribute them to other Unified HMI devices (certificates, root certificates, and CRL files).
- Export the root certificate of the Unified HMI device and its CRL file (= Certificate Revocation List) to distribute it to external communication partners.

  Note

  Alternative method

  Export the root certificate and the CRL file of the Unified HMI device using the "WinCC Unified Certificate Manager". You can find additional information here (Page 7543).

- Import the root certificate of the Unified HMI device and its CRL file separately from the remaining certificate configuration.
  The root certificate is installed by the import.

  Note

  Alternative method

  Install the root certificate and the CRL file of the Unified HMI device together with the remaining certificates configured for the device using the "WinCC Unified Certificate Manager". You can find additional information here.

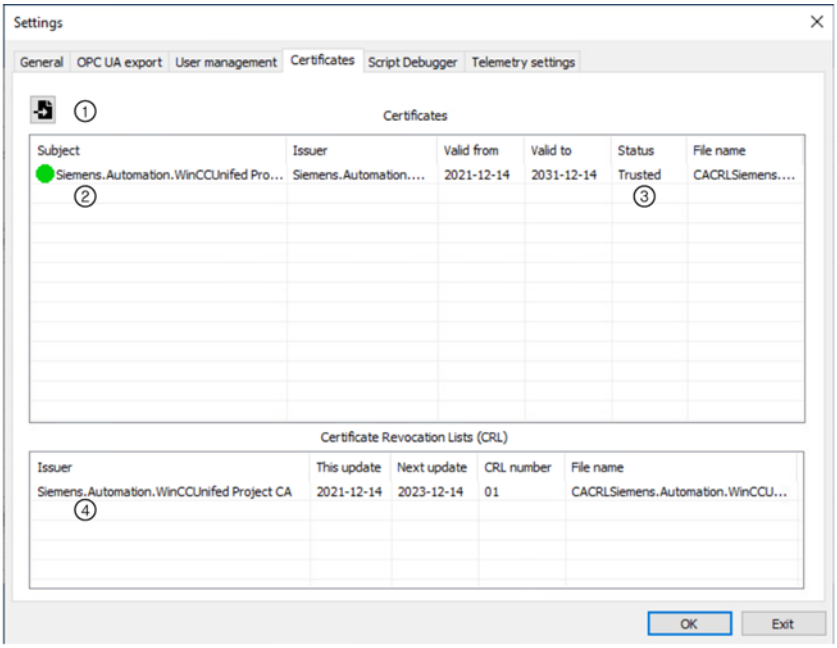
Note

Importing and exporting CRL files

The root certificate of the Unified HMI device and its CRL file must be imported separately.

When you export the root certificate of a CRL file, the CRL file is also exported. If required, you can export the CRL file separately.

Structure



① Button for importing a certificate or CRL file

By importing a certificate, you trust the certificate. You can later reject the certificate and trust it again.

② List of certificates

The following certificates are displayed:

- The root certificate installed on the Unified HMI device.
- The imported third-party certificates:
  - Application certificates (self-signed or issued by a certification authority).
  - Root certificates issued by the issuing certification authority.

③ Shows whether the HMI device trusts a certificate.

④ List of CRL files

Requirement

- The Runtime Manager is open.
- The certificates and CRL files to be imported are located in a folder to which the HMI device has access.

Managing certificates

1. Click the button in the toolbar.
2. Select the "Certificates" tab.
3. You can perform the following actions:

Action	Procedure
Import and trust	1. Click "Import new certificate or certificate revocation list (CRL)": 2. Select the location where the certificate is stored, for example, an external data carrier, and select the certificate. 3. Confirm your input. The certificate is imported and copied to the "trusted" folder on the HMI device.
Trust	Right-click on a certificate and select "Trust". The certificate is moved to the "trusted" folder on the HMI device.
Reject	Right-click the certificate and select "Reject". The certificate is moved to the "untrusted" folder on the HMI device.
Display	Right-click on a certificate and select "Show". A window with detailed information on the certificate opens.
Delete	Right-click on a certificate and select "Delete". The certificate is deleted from the certificate store on the HMI device.
Export	1. Right-click the certificate and select "Export". 2. If you have selected a root certificate, select the file format. 3. Select the target folder, for example, an external data storage medium. 4. Confirm your input. The certificate is copied to the target folder. If you have selected a root certificate, its CRL file is also copied. Distribute the files to the desired devices. To do this, proceed as described in the application help of the device.

Managing CRL files

1. Click the button in the toolbar.
2. Select the "Certificates" tab.
3. You can perform the following actions:

Action	Procedure
Import	1. Click "Import new certificate or certificate revocation list (CRL)":  2. Select the location of the CRL file, e.g. an external data storage medium, and select the file. 3. Confirm your input. The file is imported and copied to the "trusted" folder on the HMI device.
Delete	Right-click on a CRL file and select "Delete". The file is deleted from the "trusted" folder on the HMI device.
Export	1. Right-click on the CRL file and select "Export". 2. Select the file format. 3. Select the target folder, for example, an external data storage medium. 4. Confirm your input. The CRL file is copied to the target folder. Distribute the files to the desired devices. To do this, proceed as described in the application help of the device.

See also

Introduction to the WinCC Unified Certificate Manager (Page 7517)

17.4.13 Exporting tags via the OPC UA server

In the "OPC UA Export" tab, you can export the tags of the project running in Runtime via the OPC UA server into an XML file. The exported data can then be imported into another application, e.g. the TIA Portal, without the need for a connection to the OPC UA server.

The export makes it easier for you to apply an existing configuration to a new Runtime system.

You can find a detailed description in the help "OPC UA - Open Platform Communications". To do this, open the following file after installing Runtime: "<Path to the Unified installation directory>\Documentation\<Language folder>\OPCWCCU<Language code>.pdf"

For example C:\Program Files\Siemens\Automation\WinCCUnified\Documentation\English\OPCWCCUenUS.pdf

17.4.14 Activating user management

Introduction

Several projects can be loaded on one Unified PC. The configuration of their user management may differ. For a successful login to a project in Runtime, the project must be running in Runtime and the appropriate user management must be active.

In the "User administration" tab, activate the appropriate user administration. For a project with central user management, you can also adapt the connection settings to the UMC server, e.g. to add missing settings in the TIA Portal or to use different settings.

Configuring the user administration

The user management of a project is configured in the TIA Portal under "Runtime settings > User management". In the Runtime Manager, it is not possible to switch a project from local to central user management.

You can find information on configuring the user management in the TIA Portal in the TIA Portal online help.

You can find information on how to configure the runtime system settings for user management during runtime installation or later with WinCC Unified Configuration here.

Requirement

- In the Runtime system settings, it has been specified that Runtime uses the user management configuration downloaded from the TIA Portal.
- At least one user has been configured with an HMI function right for the user management active in Runtime.
- At least one user has been configured with an HMI function right for the user management that you want to activate.
- The Runtime Manager is open.
- A project is running in Runtime, and:
  - The active user management does not match the user management configured for the project.
  - For projects with central user management: The connection settings configured in the TIA Portal for the project are incomplete, or you want to use different settings.

Procedure

1. Click the button  in the toolbar.
2. Select the "User management" tab.
3. Under "Select configuration", in the "From" list, select the project whose user management configuration you want to activate in Runtime.
   Default setting after starting the Runtime Manager: The project running in Runtime

4. Confirm the confirmation prompt.
   The "Operating mode" area shows the operating mode of the user management of the selected project. The displayed options are read-only.
5. If the project selected under "From" uses local user management, click "Load user management".
   User management is activated in Runtime:
   - The user data pre-configured in the TIA Portal for the project is loaded into the local user management.
   - Runtime uses the local user management.
   - The "Status" field shows the status of the user management.

NOTICE
Possible data loss The user data configured in the TIA Portal overwrites the user data added or changed on the HMI device in the local user management. Data loss can occur.

6. If the project selected under "From" uses central user management, proceed as follows:
   - Add missing or incorrect information about the connection settings.
     By default, the identity provider address is automatically generated based on the UMC server address.
     To enter the address of the identity provider manually, deactivate the option "Generate the address of the identity provider automatically".
     To set all fields to empty, click "Reset configuration".
   - Click "Connect to server".
     The system will notify you if the configured server ID and the server ID reported during the connection attempt are different from each other. To continue with the ID reported online, click "Yes"; to continue with the configured server ID, click "No".

   User management is activated in Runtime:
   - A connection to the UMC server is established using the connection settings from the Runtime Manager.
   - Runtime uses the UMC server for user management.
   - If you later select the project under "From", the connection settings you entered are loaded.

17.4.15 Setting the Runtime Script Debugger settings

The scripts of the screens and jobs of a Runtime project can be tested using the Google Chrome script debugger.

To this end, the debugger must be configured and enabled in advance in the "Script debugger" tab in the Runtime Manager.

See also

Enabling the debugger (Page 7483)

17.4.16 Enabling telemetry service

Introduction

The telemetry service is used to analyze problems occurring in Runtime projects. It helps the Siemens support team to support you in analyzing and correcting such problems in the best possible way. The service generates an encrypted ECD file that combines the visual recording of the process running in runtime with the recording of detailed internal system information about the process.

The ECD file records the following information from the project running in runtime:

- Screen configuration of the visible screens
- User input with mouse and keyboard
- IO addresses of all connections
- Property values of the underlying CHROM objects

Note

- Enable the telemetry service only after being requested by the Siemens support team.
- The size of the ECD file depends on the length of the recording, the number of connections and the events in the process.
- The ECD file contains machine-specific and user-specific data such as user names, but not passwords.
  This data is visible to the Siemens support team. The data is not processed or stored longer than necessary.

Requirement

SIMATIC Runtime Manager has been started in admin mode.

Enabling telemetry service

1. Click the  button in the toolbar.
2. Select the "Telemetry settings" tab.
3. Under "Storage directory", specify the path to the directory where the ECD file is to be stored. Use an already existing directory.
4. Enable the "Enable telemetry" option.
5. Restart Runtime.

The telemetry service will be enabled for the currently running project.

Next steps

1. Reproduce the error scenario in Runtime.
2. Stop the telemetry service by disabling the "Enable telemetry" option and restarting Runtime.
3. Submit the ECD file to the Siemens Support team.

17.4.17 Operation via command line

The Runtime Manager has an interface with which you can start numerous functions of the Runtime Manager via a command line program:

Requirement

- Runtime and command line program are installed on the same device.
- For starting/stopping projects: Projects have been loaded into Runtime.

Procedure

1. Start the command line program.
2. Enter the command line call. Separate the individual elements of the call with spaces.
   - Enter the path to the SIMATIC Runtime Manager.exe:
     "<Runtime installation directory>\bin> start /wait SIMATICRuntimeManager.exe"
     Example: C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATICRuntimeManager.exe
   - Enter the options with which the command line program calls the Runtime Manager. The last option must be "-c".

Option	Description
`-s`	Option for starting the Runtime Manager in silent mode. Without this option, the UI of the Runtime Manager is started when the command line call is processed.
`-u`	Option to enable help messages that assist you in operating the Runtime Manager via the command line program.
`-sim`	Only use this option if you call the option "-c" with the command `projectstate`, `start`, `stop` or `remove`.
`-quiet`	Option for calling the Runtime Manager without output.
`-o`	Option for diverting the output into an Output.txt file that is stored parallel to SIMATICRuntimeManager.exe. You can redirect the output to another folder. The Unified Administrator must have write access to the folder. Example: `-o "C:\Program Files\Siemens\Automation\WinCCUnified\bin\MyOutput.txt"` If an error occurs during the write and -quiet is not set, the error indication appears on the console.
`-keepUmc`	Optional Only in combination with the `fulldownload` command Set the option to keep the Runtime UMC data.
`-overwriteUmc`	Optional Only in combination with the `fulldownload` command Set this option to replace the UMC data of the Runtime with the UMC data from the project.
`-c`	Option for inputting the commands that are transmitted to the Runtime Manager.

– After the option "-c", enter the command that the Runtime Manager should run and the argument that is transmitted to the command:

Command	Argument	Description
`start`	<Project ID>	Starts the project.
`stop`	<Project ID>	Stops the project.
`projectlist`	[ALL] or [RUNNING] Default: [ALL]	`[ALL]`: Returns a list of projects loaded in the Runtime. `[RUNNING]`: Returns the project running in Runtime.
`projectstate`	<Project ID>	Returns the state of the project running in Runtime.
`remove`	<Project ID>	Removes the project from Runtime. If the autostart option was previously set for the project: Removes the autostart option.
`securemode`	<Password>	Sets the password for secure communication with SCS. Enter the same password that Runtime uses for secure communication.
`setautostart`	<Project ID>	The project is started when the device is booted. The project must have the Project type. The option can only be set for 1 project.
`removeautostart`	<Project ID>	Removes the autostart of the project.
`fulldownload`	<Log path>	Starts the full download of a TIA Portal log. If the project is already running in Runtime, it is stopped first before the full download. To start the project after successful download, use the command `start`.
`deltadownload`	<Log path>	Starts the change loading of a TIA Portal log. Check in advance if the corresponding project is downloaded and running in Runtime.

To run multiple commands, use multiple command line calls.

3. Press Enter.

Result

- The command is executed.
- A return code with description is output in the console.
  List of possible return codes:

Return code	Description
0x00000000	Success
0x0080400b	Project already running
0x0080400c	Project started
0x0080400d	Project already stopped
0x0080400e	Project stopped
0x80000000	General error
0x80000001	Not supported (e. g. wrong command)
0x80000003	Timeout during communication with SCS
0x80000004	Invalid arguments
0x80000005	Access denied – password required for secure connection
0x8000000C	Another project is currently flagged as autostart project, remove autostart from the other project
0x80000016	Unable to connect to SCS
0x80804019	Project not found
0x80B0412E	Write output file error
0x80B0412F	Autostart option cannot be set on simulation project
0x80B04130	Empty command value
0x80B04131	archive target path could not be created
0x80B04132	project archive can not be extracted
0x80B04133	DownloadTask file can not be read
0x80B04134	Could not change UMC Data override option
0x80B04135	Missing config folder in archive
0x80B04136	Missing delta folder in archive

- An output is written to the console or to the output file.
  Requirement: The command was called without the -quiet option.

Examples

- Call a list of all projects loaded into Runtime:
  - Input: `C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATICRuntimeManager.exe -s -c projectlist [ALL]`
  - Example output:

```
[1]
 Project name:  T1
 Device name:  T1
 Project type:  Project
 Project ID:  0B527D12-6BBD-4F2F-BEB9-23E3C37A8932
 Autostart:  0
[2]
 Project name:  T2
 Device name:  T2
 Project type:  Project
 Project ID:  29DCBA1D-C615-4560-AFB4-94EB9565682C
 Autostart:  0
[3]
 Project name:  T3
 Device name:  T3
 Project type:  Project
 Project ID:  96FE68D0-5337-4072-A96C-F7C1D7525CAF
 Autostart:  0
```

- Call the project running in Runtime:
  Input:
  `C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATiCRuntimeManager.exe -s -c projectlist RUNNING`
- Query project state:
  Input:
  `C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATICRuntimeManager.exe -s -c projectstate 96FE68D0-5337-4072-A96C-F7C1D7525CAF`
- Start a project:
  Input:
  `C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATICRuntimeManager.exe -s -c start 96FE68D0-5337-4072-A96C-F7C1D7525CAF`
- Stop a project:
  Input:
  `C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATICRuntimeManager.exe -s -c stop 96FE68D0-5337-4072-A96C-F7C1D7525CAF`
- Remove a project from Runtime:
  Input:
  `C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait SIMATICRuntimeManager.exe -s -c remove 96FE68D0-5337-4072-A96C-F7C1D7525CAF`

- Example of a query regarding the state of a simulation project:
  Input:
  ```
  C:\Program
  Files\Siemens\Automation\WinCCUnified\bin>SIMATICRuntimeManager.ex
  e -s -sim -c projectstate 96FE68D0-5337-4072-A96C-F7C1D7525CAF
  ```
- Set password for secure communication with Runtime:
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -c securemode <password>
  ```
- Enable autostart for a project:
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -c setautostart 28AC5BD5-0741-42D1-
  B3C6-503359F32B7E
  ```
- Disable autostart for a project:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -c removeautostart
  28AC5BD5-0741-42D1-B3C6-503359F32B7E
  ```
- Perform a full download of a TIA Portal log:
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -c fulldownload
  "C:\Users\admin\Desktop\ HMI_RT_1[Project1] - Full 2019-10-21 -
  08.00.22.zip"
  ```
- Download only the changes of a TIA Portal log (delta download):
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -keepUmc -c fulldownload
  "C:\Users\admin\Desktop\HMI_RT_1[Project1] - Full 2020-03-27 -
  11.39.51.zip"
  ```
- Retain UMC data during full download:
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -c deltadownload
  "C:\Users\admin\Desktop\ HMI_RT_1[Project1] - Delta 2019-10-21 -
  08.03.18.zip"
  ```
- Replace UMC data during full download:
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s -overwriteUmc -c fulldownload
  "C:\Users\admin\Desktop\HMI_RT_1[Project1] - Full 2020-03-27 -
  11.39.51.zip"
  ```
- Enable help messages:
  Input:
  ```
  C:\Program Files\Siemens\Automation\WinCCUnified\bin> start /wait
  SIMATICRuntimeManager.exe -s – u
  ```

See also

Enabling the debugger (Page 7483)

17.5 Certificate Manager

17.5.1 Basics

17.5.1.1 Introduction to the WinCC Unified Certificate Manager

Protection through certificates

Communication within the plant must be protected to secure plants, systems and networks against cyber threats. Security-relevant components of WinCC Unified devices therefore use encrypted communication protocols.

It is recommended to use certificates created with a certificate authority for this communication. In this case, each communication partner is assigned a unique certificate that is used for authentication and encryption.

Task of the WinCC Unified Certificate Manager

With WinCC Unified Certificate Manager you create a certificate authority. Then, with the certificate authority, create the certificates of the Unified devices and distribute them.

On Unified PCs, you also install the certificates via Certificate Manager. On Unified Comfort Panels, use the Control Panel to do this.

Note

No support for certificates from external certification authorities

The use of the Certificate Manager is required to issue certificates for Unified Runtime via a certificate authority.

Functions

The Certificate Manager offers you the following functions for the creation, distribution and installation of the certificates:

- Central creation of certificates for Unified devices in the network
- Creation of a certificate authority

- Creation of the application certificates for the following WinCC Unified components:
  - WinCC Unified Runtime (Webserver (IIS))
  - WinCC Unified OPC UA server
  - WinCC Unified OPC UA exporter
  - WinCC Unified OPC UA client
  - WinCC Unified Runtime Collaboration
  - WinCC Unified Audit Trail system
- Renewing existing certificates
- Encrypted export of the certificates for manual distribution to the Unified devices
- Encrypted import and installation of certificates on Unified PCs
- Encrypted export and import of the root certificate, CRL file, and private key, as well as all device certificates for data backup and recovery.
- Export of the root certificate and its CRL file for distribution to external communication partners
- Export an updated CRL file for distribution to Unified devices and external communication partners.

Additional information

You can find more information about the deployment of certificates under "How to use certificates" in the user help "SIMATIC Unified PC Installation".

17.5.1.2 Certificate authority

Total configuration of the certificate authority

The total configuration of the certificate authority (CA) includes:

- The private key
  The certificate authority uses it to sign the application certificates. The private key remains on the certificate authority device.
- The public key/root certificate (CA certificate) and a CRL file (Certificate Revocation List).
  Root certificate and CRL file are distributed to the Unified devices and their external communication partners.

  Note

  This help uses the runtime root certificate.

- The devices added to the certificate authority and their application certificates.

CA container

The CA container (Certificate Authority container) includes:

- The root certificate
- The CRL file
- The application certificates of all devices added to the certificate authority

17.5.1.3 Required certificates

Introduction

You need the following certificates to use a Unified device:

- Root certificate of the certificate authority and its Certificate Revocation List (CRL file)
- The application certificates issued by the certificate authority

These certificates form the certificate configuration of the Unified device.

Root certificate and CRL file

The root certificate of the certificate authority and its CRL file must be distributed to the Unified devices and the external communication partners.

The trust relationship between a Unified device and its communication partners is established by the devices trusting each other's root certificates.

Unified devices whose certificates come from the same certificate authority automatically trust each other after installing their certificate configuration.

Application certificates

The device requires the following application certificates:

- For accessing WinCC Unified Runtime via a web page:
  The device with the Unified Runtime installation requires a web server certificate.

  Note

  A web server (IIS) must also be installed on the device.

  The web clients can access the runtime web page only if the root certificate of the web server certificate on the clients has been previously configured once as trusted in the browser.
  See Installing root certificate for access via web client (Unified PC) (Page 7545).

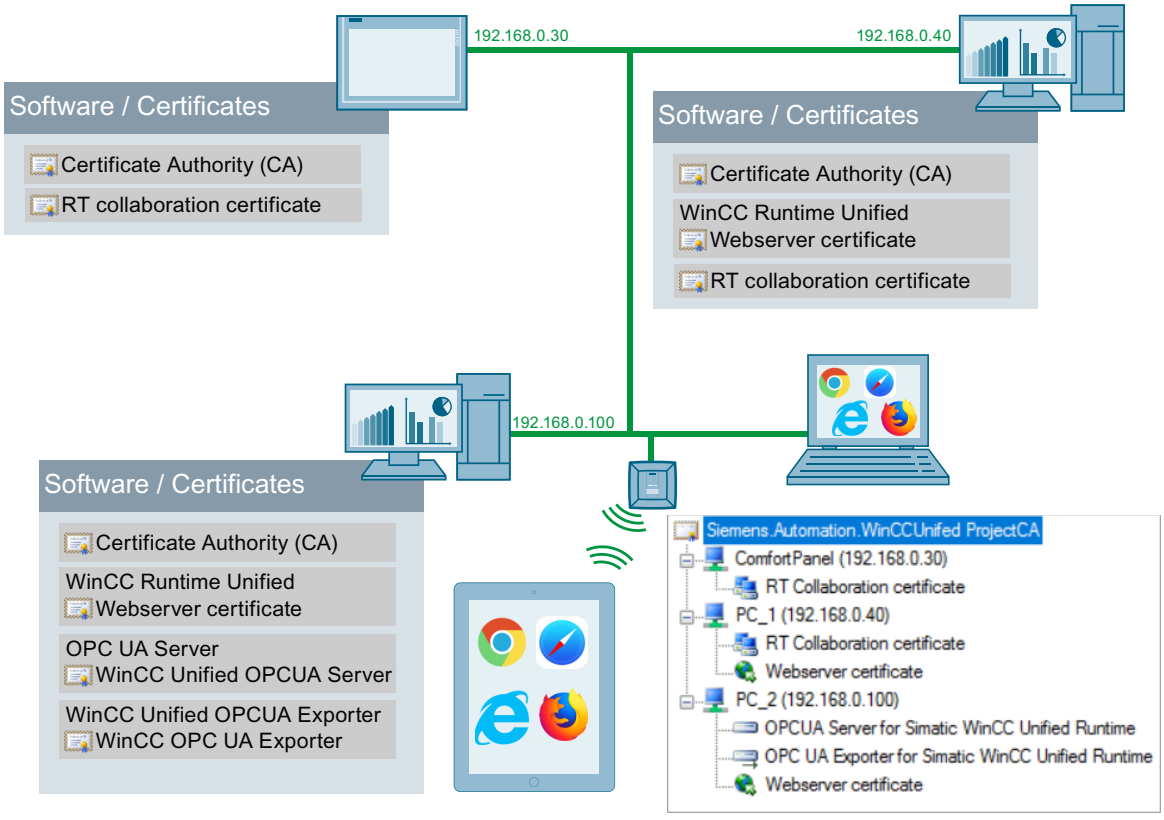
- For data exchange between Unified devices (WinCC Unified Collaboration):
  The devices require a Runtime Collaboration certificate.

- For OPC UA communication between devices:
  - A Unified device used as OPC UA server requires an OPC UA Server certificate and an OPC UA Exporter certificate.
    Alternatively, you can use the self-signed default certificates generated for the OPC UA server and OPC UA exporter.
  - A Unified device that is to communicate with an OPC UA server requires an OPC UA client certificate.
- For GMP-compliant use of a unified device:
  The device requires an Audit Trail System certificate.

Example

Two Unified PCs and one Unified Comfort Panel are operated in the following example:

- The PCs provide web pages for runtime visualization. Therefore, you need a web server certificate.
- A PC (192.168.0.100) runs an OPC UA server. Therefore it requires the Unified OPC UA Server and OPC UA Exporter certificates.
- The panel (192.168.0.30) and a PC (192.168.0.40) should exchange runtime data. Both devices require a Runtime Collaboration certificate.
- The root certificate is installed and classified as trustworthy on all devices.
- When accessing the Runtime web pages from external devices via a browser, the root certificate must be installed in the browser's certificate store. The Unified Runtime start page provides a corresponding download link.



See also

Exporting root certificate and CRL file (Page 7543)

17.5.1.4 Password requirements

The passwords defined in the Certificate Manager must meet the following requirements:

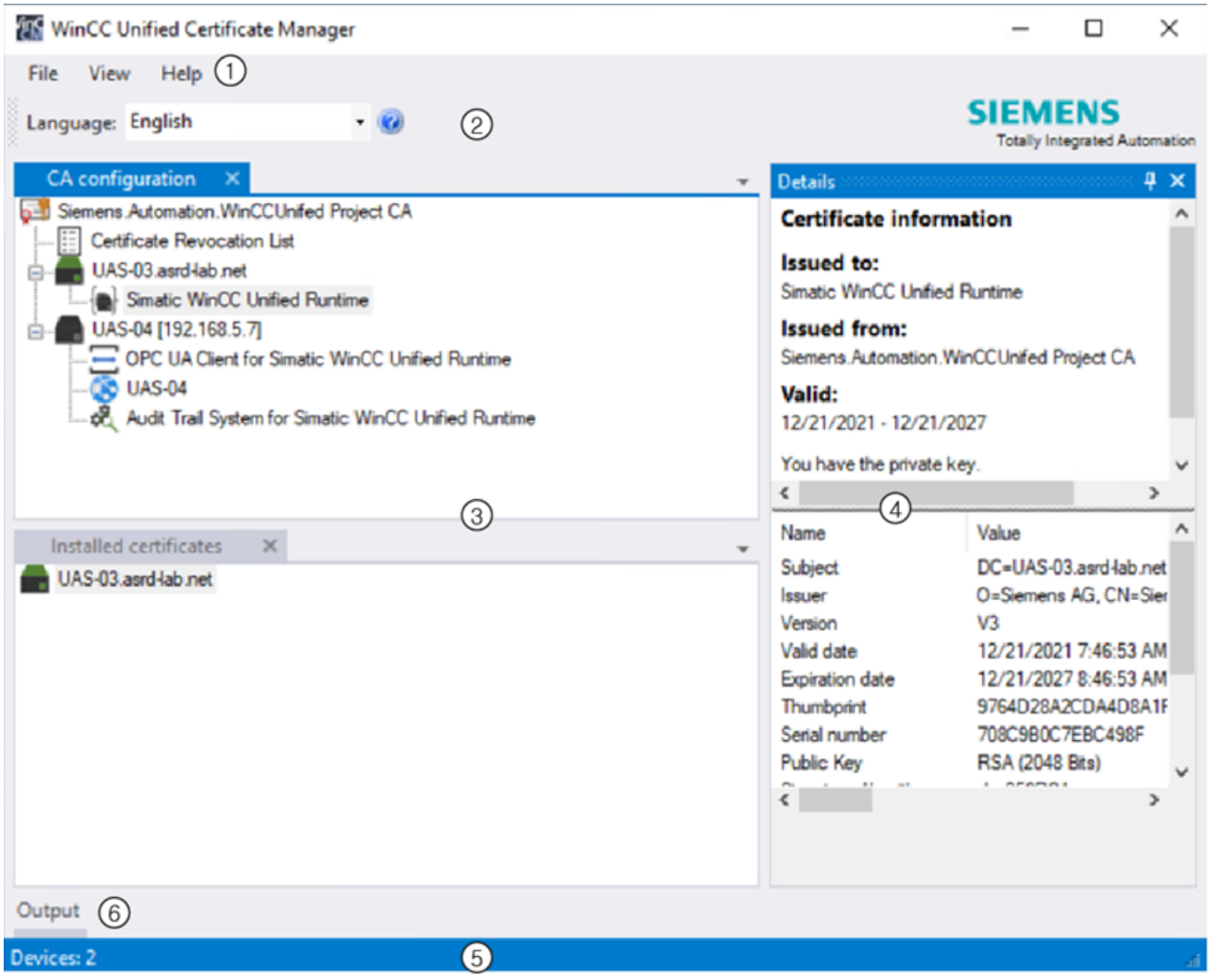
- Length: At least 8 characters
- In each case at least one uppercase letter, one lowercase letter, one number and one special character

17.5.2 Certificate Manager interface

17.5.2.1 Structure of the user interface

Overview

The interface of WinCC Unified Certificate Manager has the following structure:



① Menu bar

② Toolbar

③ Work area with the "CA configuration" and "Installed certificates" tabs

④ "Details" area (fixed)

The "Details" area shows you detailed information about the certificate selected in the work area.

⑤ Information bar
⑥ "Output" area (hidden)
The "Output" area logs operator control actions.

You can customize the display of the interface to suit your needs. See also Customize surface (Page 7525).

Menu bar

Menu	Description
"File > Exit"	Closes Certificate Manager.
"View"	Configure which Certificate Manager interface elements you see. You can open or close the following interface elements: • "Output" area • "Details" area • "CA configuration" tab • "Installed certificates" tab
"Help"	"Certificate Manager Help" Opens the user help in a browser.
	"Info Certificate Manager" Opens a dialog with information about the installed software version.

Toolbar

Button	
	To change the user interface language
	To call the user help

Tab of the working area

See "CA configuration" tab (Page 7523) and "Installed certificates" tab (Page 7525).

17.5.2.2 "CA configuration" tab

On a certificate authority device

On a certificate authority device, you create and configure the certificate authority in the "CA configuration" tab:

- You create the certificate authority and its root certificate.
- You add devices.
- You create the application certificates of the devices.

- You carry out exports:
  - To make the certificates available on other devices
  - To backup data
- You recreate certificates.
  You have the following options:
  - Recreating a root certificate
  - Updating a CRL file
  - Recreating a certificate configuration of a device
  - Recreating a single application certificate of a device
- When the certificate authority device is used as a Unified PC: You install the application certificates of the device.

Note

Content of the tab

After starting Certificate Manager, you will see the same data that the certificate authority had when you last closed Certificate Manager:

- If no data has been generated yet, you will see the nodes "Open configuration ..." and "Create certificate authority ..."
- If data has already been generated, you will see the root certificate and its CRL file, as well as the configured Unified devices and their application certificates.
  You can edit them.

On standard Unified PCs

On Unified PCs that do not serve as a certificate authority, you perform the following actions in the tab:

- You import the certificate configuration of the device
- You install the entire certificate configuration or a single application certificate
- You delete installed certificates

Note

Content of the tab

After launching Certificate Manager, you will see the nodes "Open new configuration ..." and "Create certificate authority ..."

After opening a new configuration, you will see the root certificate of the certificate authority and its CRL file, as well as the Unified devices configured at the certificate authority and their application certificates.

You can install only the certificate configuration of the local device. The display of the certificate configuration of the other devices is for information purposes. You cannot change their certificate configuration.

Closing Certificate Manager also closes the configuration.

See also

Structure of the user interface (Page 7522)

17.5.2.3 "Installed certificates" tab

In the "Installed certificates" tab, you can see which application certificates are installed on the local device.

You have the option to uninstall certificates by deleting them.

See also

Structure of the user interface (Page 7522)

17.5.2.4 Customize surface

Display and arrangement of the interface elements of WinCC Unified Certificate Manager are configurable:

User interface elements	Close / Open	Move	Undock / dock	Fix / Unfix	Show / Hide
"Details" area	✓	✓	✓	✓	✓
"Output" area	✓	✓	✓	✓	✓
Tab of the working area	✓	✓	-	-	-

Closing and opening

To close a user interface element, click the "X" button. Alternatively, disable it in the "View" menu.

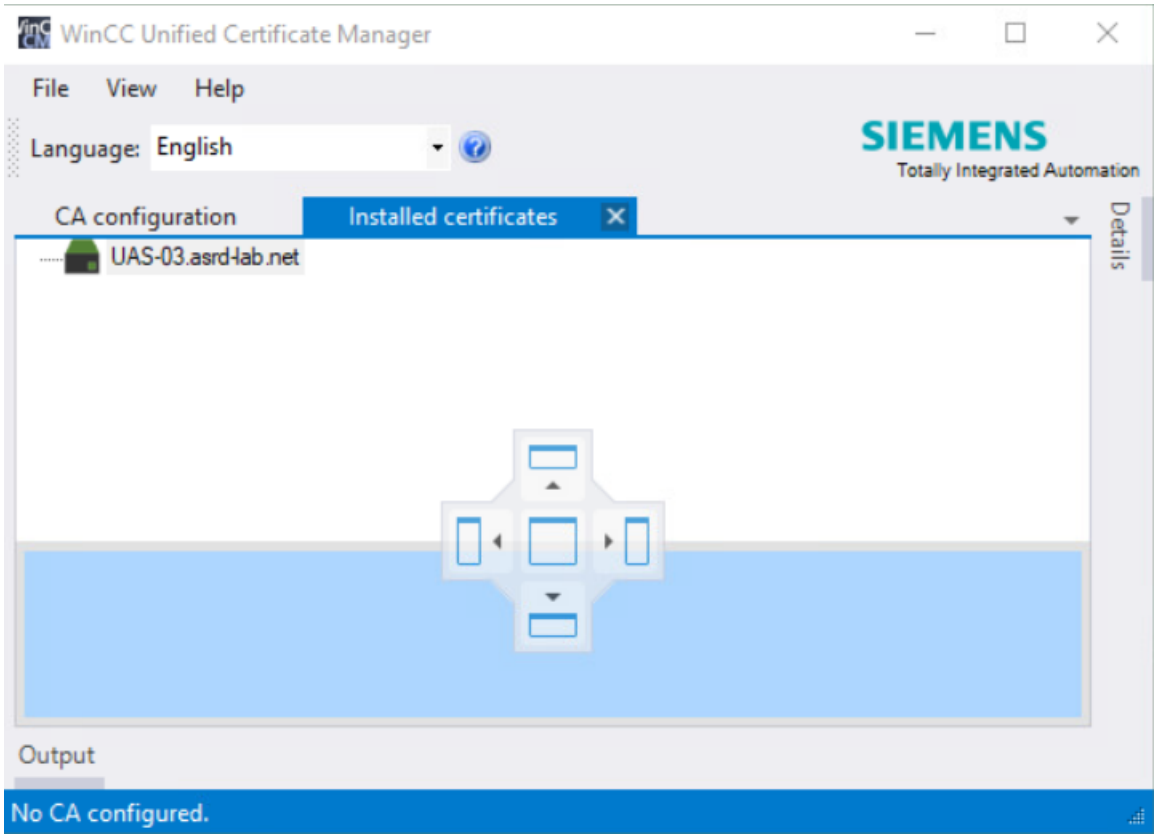
To open a closed user interface element, enable it in the "View" menu.

Move

1. Move the title bar of the desired user interface element with the left mouse button pressed. Possible insertion positions are displayed in the interface:

   The offered insertion positions depend on which element you move and which elements the application window already displays.

2. To see a preview of the new arrangement, move the mouse cursor to one of the positions and keep the mouse cursor pressed:



3. Release the mouse cursor over the desired insertion position.

The user interface element is moved.

Undocking and docking

When you move the header of the "Details" or "Output" area, the area is undocked from the application window and displayed as a standalone window. You can move the window freely.

To dock the area back to the application window, move it to one of the suggested insertion positions.

Fixing and unfixing

The following button fixes or unfixes the "Details" and "Output" areas:

Representation of the user interface	Status	Changing setting
	Fixed The area is displayed even if it does not have the focus.	Click the button to switch the setting.
	The area is hidden as soon as it loses focus.	

Showing and hiding

Requirement

The "Details" and "Output" areas are not fixed.

Procedure

To show an area, click on its text. The area is displayed.

It is automatically hidden when you click the mouse cursor outside the area.

17.5.2.5 Changing the user interface language

Procedure

1. Click the button with the arrow in the toolbar:

2. Select the desired language.

Result

Changing the user interface language

17.5.3 Making certificates available

This section describes which steps are required to provide certificates with WinCC Unified Certificate Manager.

Generating a certificate configuration

On the device that serves as the certificate authority, do the following.

1. Create the certificate authority.
   This creates the root certificate, CRL file and private key.
   See Creating a certificate authority and root certificate (Page 7530).
2. Add the required Unified devices to the certificate authority.
   See Adding devices (Page 7531).
3. Create the desired application certificates for the devices.
   See Add application certificates (Page 7533).

Exporting the certificate configuration

After you complete the certificate configuration of the devices, export the certificate configuration.

The procedures differ for PC devices and panel devices. See Exporting certificate configuration (Unified PC) (Page 7536) and Exporting the certificate configuration (UCP) (Page 7540).

Note

Distribution of the root certificate and CRL file

Root certificate and CRL file are part of the certificate configuration of a device. They are exported or imported with the certificate configuration, installed and classified as trusted.

Certificate Manager also offers the option of exporting the root certificate and CRL file individually, e.g. to distribute them to external communication partners.

On Unified PCs, you can also export the root certificate and CRL file via SIMATIC Runtime Manager. You can find additional information here (Page 7504).

For an overview of all Certificate Manager export options, see Export options (Page 7535).

Making certificates available on the Unified devices

To make the certificates of a Unified HMI device available on the device, follow these steps:

1. Import the certificate configuration.
   See Importing certificate configuration (Unified PC) (Page 7538) and Importing and installing certificate configuration (UCP) (Page 7541).
2. Install the entire certificate configuration or individual certificates.
   See Installing certificates (Unified PC) (Page 7539) and Importing and installing certificate configuration (UCP) (Page 7541).

The procedures differ for PC devices and panel devices.

Depending on the device, use the following tool:

- Unified Comfort Panel: Control panel > "Security" function
- Unified PC: WinCC Unified Certificate Manager

Trust communication partners

For successful communication, the Unified device must trust the root certificates of its communication partners and vice versa.

Trust relationship between Unified devices

Unified devices whose certificate configuration comes from the same certificate authority automatically trust each other after the certificate configuration is installed.

Relationship of trust with external communication partners

To establish the trust relationship with an external communication partner, follow these steps:

1. On the certificate authority device or on the Unified device:
   Export the root certificate and CRL file to an external data storage medium.
   See Exporting root certificate and CRL file (Page 7543).
2. On the external communication partners:
   - Connect the external communication partner to the external data storage medium.
   - Copy the files and trust them. To do this, proceed as described in the user help of the device.
   - Export the root certificate of the external communication partner and its CRL file to the external data storage medium.
3. Connect the external data storage medium to the Unified device.
4. Import the root certificate and CRL file of the external communication partner into the Unified device and trust it:
   - Unified PC: Use SIMATIC Runtime Manager.
   - Unified Comfort Panel: In the control panel of the device, under "Security", use the "Certificates" > "Import" function.

During the next connection attempt, the communication partners mutually accept their application certificates.

Recreating certificates

You have the option to recreate certificates.

See Recreating certificates (Page 7551).

See also

Importing and installing certificates manually (UCP) (Page 7542)

17.5.4 Creating a certificate authority and root certificate

Requirement

On the Unified PC that is to serve as a certificate authority, no certificate authority is created in WinCC Unified Certificate Manager.

Procedure

1. On your network, select the Unified PC that will be used as the certificate authority.
2. Open WinCC Unified Certificate Manager on this device.
3. Select the "CA configuration" tab.
4. In the work area, double-click "Create new certificate authority".
5. Enter the properties of the root certificate in the "New certificate authority" dialog. The fields are freely editable.
   Mandatory fields:
   - "Name"
   - Password fields for the private key
     See also Password requirements (Page 7521).

   If necessary, select a different cryptographic key length and runtime for the certificate.
6. Click "Create".

Result

- The private key is generated.
- The root certificate is generated.
- An empty CRL (Certificate Revocation List) file is generated.
- In the "CA configuration" tab, a node for the root certificate is created and below it one for the CRL file.

Note

The private key is only available on the certificate authority device. The certificate authority uses it to sign the application certificates of the Unified devices.

The root certificate and the CRL file are part of the certificate configuration of the Unified devices. They are exported or imported when the certificate configuration is exported or imported. During the installation of the certificate configuration on the device, they are automatically installed and classified as trusted.

Next steps

- Add the Unified devices to the root certificate and create their application certificates. See Adding devices (Page 7531) and Add application certificates (Page 7533).
- To distribute the Unified root certificate and its CRL file without the certificate configuration of the devices, for example, to external communication partners, export them individually. See Exporting root certificate and CRL file (Page 7543).

Deleting certificate authority and root certificate

NOTICE
Data loss prevention Delete the certificate authority and root certificate only in the following cases: • After you have saved the data of the certificate authority. • When you no longer need the certificate authority and its data.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Click the root certificate on the left and select "Delete".

Result

The certificate authority and all its data will be deleted from the device.

Note

If the certificate configurations were already installed on the Unified devices, the certificates are still installed there. Delete them there if necessary.

17.5.5 Adding devices

Requirement

A certificate authority has been created in WinCC Unified Certificate Manager on the Unified PC that is used as the certificate authority.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
   You see the root certificate and its CRL file, as well as all the devices that have already been added and their application certificates.

3. Right-click the root certificate and select "Add device ...".
4. In the "New device" dialog, enter the device name, the IP address of the device, or both. When creating application certificates for this device, these entries are inserted into the certificates and used for the validation.

Note

Allowed device names

Either the host name or the fully qualified domain name can be used as the device name.

The use of the name "localhost" is not allowed and will be automatically replaced by Certificate Manager with the device name of the local device.

Note

Required inputs

- For devices in a domain, enter the fully-qualified domain name as the device name. This avoids validation errors when accessing the web pages.
- If the identity provider and Unified Runtime website are addressed by IP address, enter the IP address.
- For devices that are used as OPC UA servers, enter the device name.
- For devices with dynamic IP addresses, enter only the device name.

Result

A node for the device is generated in the "CA configuration" tab.

Icons of the device nodes:


The local machine (if added)


Other devices

Next step

Create the application certificates of the added device.

Delete device

Procedure

To delete a device, click on the device on the left and select "Delete".

Result

The device and its application certificates are deleted from the certificate authority.

Note

Deleting does not affect the certificate configuration installed on the device.

If necessary, delete the certificates from the device. On a Unified PC, you use Certificate Manager to do this. On a Unified Comfort Panel, use the "Security" function in the Control Panel.

Handling a device after changing the IP address or the computer name

If the IP address or the computer name of a device added in Certificate Manager is changed later, do the following:

1. Delete the device in Certificate Manager.
2. Create it again using the new IP or the new computer name.
3. Add the desired application certificates to the device.
4. Export the certificate configuration and distribute it.

See also

Creating a certificate authority and root certificate (Page 7530)

Add application certificates (Page 7533)

17.5.6 Add application certificates

Requirement

A device has been added to the certificate authority in WinCC Unified Certificate Manager.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Right-click on a device and select "Add <Certificate type> ..."

4. Enter the properties of the certificate in the "New certificate" dialog.
   If necessary, select a different cryptographic key length and runtime for the certificate.

   Note

   Runtime

   For web server certificates, the runtime is limited to a maximum of 27 months. Longer runtimes are not accepted by some browsers.

   Note

   Use the "Fully qualified domain name" as name for web server certificates.

5. Click "Create".
6. Repeat the last three steps until the device has the required application certificates.

Result

The certificate configuration of the device is completed.

Next step

Export the certificate configuration.

Note

If the certificate authority device is used as a Unified device, no export is necessary. You can install the certificate configuration or individual certificates directly.

Deleting an application certificate

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Click on the application certificate under the desired device on the left and select "Delete".

Result

The application certificate is deleted.

Note

Deleting does not affect the certificate configuration installed on the device.

If necessary, delete the certificate from the device. On a Unified PC, you use Certificate Manager to do this. On a Unified Comfort Panel, use the "Security" function in the Control Panel.

Recreating an application certificate

You have the option to recreate application certificates, for example, when their runtime ends.

See section Recreating application certificates (Page 7553).

See also

Exporting certificate configuration (Unified PC) (Page 7536)

Exporting the certificate configuration (UCP) (Page 7540)

Adding devices (Page 7531)

17.5.7 Export options

The following table provides an overview of which export options WinCC Unified Certificate Manager offers and how to use them:

Option	Use	Available on the certificate authority device	Available on the Unified target device (Unified PC)
Export of the CA container	To provide the certificate configuration from one or more Unified PCs.	✓	-
Exporting the certificate configuration of a device	To make available the certificate configuration of a single Unified Comfort Panel. Can also be used to make available the certificate configuration of a single Unified PC.	✓	-
Exporting a single application certificate	To make available as a public certificate	✓	✓
Exporting the root certificate and CRL file	To establish the trust relationship between a Unified device and its external communication partners. To make available an updated CRL file	✓	-
Exporting the complete configuration of the certificate authority	To backup data of the certificate authority	✓	-

17.5.8 Exporting, importing and installing for Unified PCs

17.5.8.1 Exporting certificate configuration (Unified PC)

Introduction

Exporting the certificate configuration of a Unified PC is required in the following cases:

- After adding the device and configuring its application certificates for the first time
- After adding, deleting or recreating application certificates
- After the recreation of the root certificate
- After updating the CRL file

Note

If you only want to update the CRL file, exporting the certificate configuration is not mandatory. You can also export the CRL file individually and import it to the PC device via SIMATIC Runtime Manager.

Note

Exporting the certificate configuration is not necessary if the certificate authority device is used as a Unified HMI device and you only want to provide the modified certificate configuration of this device.

In this case, you can install the certificate configuration or individual certificates directly without exporting and importing them first.

Export options

To export the certificate configuration of a Unified PC, you can select from the following options:

- Export CA container
  The certificate configurations of all devices are exported.
- Export device
  Export only the certificate configuration of a specific Unified PC.

Note

In both cases, you can only install the certificate configuration of this PC on the Unified PC.

	Tip for an efficient procedure
Recommended procedure: • The certificate configuration of multiple devices was changed: Export the CA container. • The certificate configuration of a single Unified PC was changed: Export the certificate configuration of this device.	

Requirement

The certificate configuration of a Unified PC is completed.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Follow these steps:
   To export the certificate configurations of all devices:
   - Right-click on the root certificate.
   - Select "Export > CA container ...".

   To export only the certificate configuration of the Unified PC:
   - Right-click on the Unified PC.
   - Select "Export device > To PC".
4. Enter and repeat a password in the "Export" dialog to protect the export file.
   See also section Password requirements (Page 7521).
5. Click "Export".
6. Click on "Save" and select the storage location and the file name.

Result

The certificate configuration of the Unified PC or the certificate configurations of all devices are stored encrypted with the specified password in a secure storage file:

Next step

Import the certificate configuration to the device and install it.

See also

Importing certificate configuration (Unified PC) (Page 7538)

Add application certificates (Page 7533)

Installing certificates (Unified PC) (Page 7539)

17.5.8.2 Importing certificate configuration (Unified PC)

Note

Importing the certificate configuration is not necessary if the certificate authority device is used as a Unified HMI device, and you only want to provide the modified certificate configuration of this device.

In this case, you can install the certificate configuration or individual certificates directly without exporting and importing them first.

Requirement

- On the certificate authority device, the CA container or the certificate configuration of the device was exported.
- The Unified PC whose certificate configuration you want to import has access to the storage location of the export file.

Procedure

1. Open the WinCC Unified Certificate Manager on the Unified PC.
2. Select the "CA configuration" tab.
3. Double-click "Open configuration ...".
4. Select the export file.
5. Enter the password selected during export.
6. Confirm your entries.

Result

The configuration file is loaded to the "CA configuration" tab. The content of the tab depends on the selected export option:

- CA container exported:
  You will see the certificate configurations of all devices of the certificate authority.
  You can install only the certificates of the local device. The display of the other devices is for information purposes. You cannot change their configuration.
- Certificate configuration of the device exported:
  Display and installation are limited to the certificate configuration of the local device.

Exiting Certificate Manager closes the loaded configuration.

Next step

Install the certificates on the device.

See also

Exporting certificate configuration (Unified PC) (Page 7536)

Installing certificates (Unified PC) (Page 7539)

17.5.8.3 Installing certificates (Unified PC)

You have the option to install the entire certificate configuration of the device or individual application certificates.

Requirement

- WinCC Unified Certificate Manager is open on the Unified PC on which you want to install the certificates.
- The certificate configuration of the device has been imported to the device using Certificate Manager.
- The "CA configuration" tab is visible.

Procedure

Select one of the following options:

- To install the entire certificate configuration of the device:
  - Right-click the node of the local machine.
    The local machine has the following icon: 
  - Select "Install all certificates".
- To install a single application certificate of the device:
  - Under the local machine node, right-click the certificate.
  - Select "Install".

Result

Depending on what you have selected, the entire certificate configuration or the individual application certificate is installed.

The following applies:

- The application certificates are installed in the certificate stores defined for the respective application.
- The root certificate is trusted in each certificate store.
- The CRL file is installed in the designated certificate store.
- When a web server certificate is installed, it is automatically linked to the WinCC Unified website. The web server certificate replaces the certificate, if any, selected during the Runtime installation.
  The web page will then be restarted to enforce the use of the new certificate. Any connected web browsers will be disconnected as a result and will have to log in again.

Note

The following certificates only become effective after a restart of the WinCC Unified Runtime:

- OPC UA server certificate
- Runtime Collaboration certificate
- Audit Trail system certificate

See also

Importing certificate configuration (Unified PC) (Page 7538)

Exporting certificate configuration (Unified PC) (Page 7536)

17.5.9 Export, import and installation for Unified Comfort Panels

17.5.9.1 Exporting the certificate configuration (UCP)

Introduction

Exporting the certificate configuration of a Unified Comfort Panel is required in the following cases:

- After adding the device and configuring its application certificates for the first time
- After adding, deleting or renewing individual application certificates
- After the recreation of the root certificate
- After updating the CRL file

Note

Alternatively, you can export the CRL file individually and import it into the Control Center of the Unified Comfort Panel using the "Security" function.

Requirement

The certificate configuration of the device is completed.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Click on the desired device on the right and select "Export device > To Panel".

4. In the dialog that opens, enter and repeat a password to protect the export file.
   See also section Password requirements (Page 7521).
5. (Optional) Adjust the number of iterations for the encryption.
6. Click "Export".
7. Select the file path and file name and click "Save".
   The data is stored in a TAR log and encrypted with the password.
8. Copy the TAR log to an external storage medium.

Next step

Import the certificate configuration to the device and install it.

See also

Add application certificates (Page 7533)

Importing and installing certificate configuration (UCP) (Page 7541)

Importing and installing certificates manually (UCP) (Page 7542)

17.5.9.2 Importing and installing certificate configuration (UCP)

Requirement

- On the certificate authority device, the certificate configuration of the Unified Comfort Panel was exported to a TAR log via "Export device > To Panel".
  See section Exporting the certificate configuration (UCP) (Page 7540).
- The TAR log was copied to an external storage medium.

Procedure

1. Connect the panel to the external storage medium.
2. Import the certificate configuration to the device.
3. Install the certificates by trusting them.

For the import and the installation, use the "Certificates" > "Import" function under "Security" in the control panel of the device.

You can find a detailed description in the "SIMATIC HMI devices Unified Comfort Panels" operating instructions.

Note

Manual importing and installing

Alternatively, you can import the certificates to the Panel manually and install them. See section Importing and installing certificates manually (UCP) (Page 7542).

17.5.9.3 Importing and installing certificates manually (UCP)

You can also import and install the certificate configuration of a Unified Comfort Panel manually instead of using the control panel of the device.

Requirement

- The certificate configuration of the Unified Comfort Panel was exported to a TAR log via "Export device > To Panel".
  See section Exporting the certificate configuration (UCP) (Page 7540).
- The TAR log was copied to an external storage medium.

Procedure

1. Decrypt the export file using OpenSSL.
   `openssl enc -d -aes256 -salt -iter <25000> -in <exportfilename> -out <tarfilename.tar> -k <password>`
   The value for the parameter `-iter` must match the iteration count specified during export.
   The decrypted TAR log contains the configured certificates in the respective application-specific folder structure.
2. Copy the file to the Panel device.
3. Manually distribute the certificates to the specific repositories of the respective application.

17.5.10 Exporting a single application certificate

With WinCC Unified Certificate Manager, you can export application certificates individually as public certificates.

Requirement

An application certificate has been added to a device in WinCC Unified Certificate Manager.

Exporting certificate to the certificate authority device

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Right-click on the application certificate under the device.
4. Select "Export certificate ...".
5. Select a file format.
6. Confirm your entries.
7. Select a target folder.
8. Confirm your entries.

Result

The public key of the certificate is exported. Distribute it to the external communication partners.

Export certificate on the device

Additional requirements

- The device is a unified PC.
- The application certificate has been imported into the device.

Procedure

1. On the Unified PC, open Certificate Manager.
2. Select the "Installed certificates" tab.
3. Right-click on the application certificate.
4. Select "Export certificate ...".
5. Select a file format.
6. Confirm your entries.
7. Select a target folder.
8. Confirm your entries.

See also

Add application certificates (Page 7533)

Importing certificate configuration (Unified PC) (Page 7538)

17.5.11 Exporting root certificate and CRL file

Introduction

With WinCC Unified Certificate Manager you can export and distribute root certificate and CRL file separately from the certificate configuration, as a public certificate. This is necessary in the following cases, for example:

- To establish the trust relationship between a Unified device and its external communication partners.
- To update an expired CRL file on a Unified device.

You can select between the following options:

- Exporting root certificate and CRL file
- Export the CRL file only

Note

Alternative path for Unified PCs

If the Unified device is a PC on which the root certificate and its CRL file are already installed, you can also export both files via SIMATIC Runtime Manager.

You can find additional information here (Page 7504).

Requirement

A certificate authority has been created on the certificate authority device in Certificate Manager.

Exporting root certificate and CRL file

1. On the certificate authority device, open Certificate Manager.
2. In the "CA configuration" tab, click the root certificate on the right.
3. Select "Export > CA Certificate ...".
4. Select a file format.
5. Confirm your entries.
6. Select a target folder.
7. Confirm your entries.

The root certificate and its CRL file are exported to the target folder, each to a separate file.

Export CRL file only

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Under the root certificate, right-click the Certificate Revocation list.
4. Select "Export".
5. Select a file format.
6. Confirm your entries.
7. Select a target folder.
8. Confirm your entries.

The CRL file is exported to the target folder.

Distribute files

After the export, distribute the files to the target devices:

- On a Unified PC you install the files with SIMATIC Runtime Manager.
- On a Unified Comfort Panel, install the files in the Control Panel under "Security" using the "Certificates" > "Import" function.
- To distribute files to external communication partners, proceed as described in the user help for the device.

See also

Creating a certificate authority and root certificate (Page 7530)

17.5.12 Installing root certificate for access via web client (Unified PC)

Using web certificates

To enable web browsers to establish a secure connection to WinCC Unified, the root certificate with which the web server certificate of WinCC Runtime was issued must be known in the web browser as a trusted certification authority.

By installing the web server certificate on the PC device, the public root certificate is made available as a download for installation in web browsers on the WinCC Unified home page.

The procedure for installing the root certificate differs depending on your web browser.

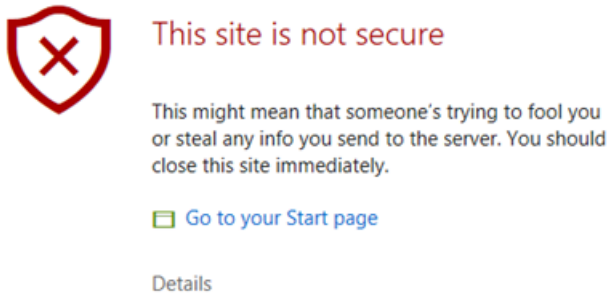
Installing the root certificate for Chrome and Microsoft Edge

Chrome and Microsoft Edge use the Windows system certificate store.

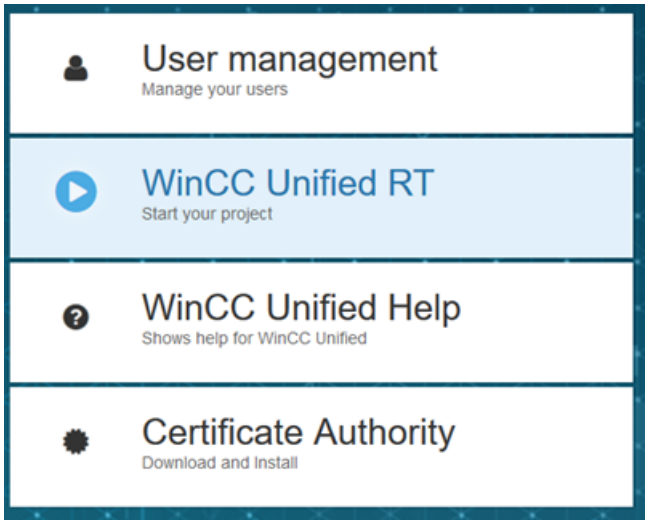
- On devices **with WinCC Unified installation** that have been configured with the Certificate Manager, these web browsers can immediately establish a secure connection to the WinCC Unified web pages because the root certificate has already been installed in the system certificate store.
- On devices **without WinCC Unified Installation** the root certificate must be installed manually.

To install manually, follow these steps (for example, Microsoft Edge):

1. Open the WinCC Unified home page via the URL https://<host name>
   At first, an error message appears:

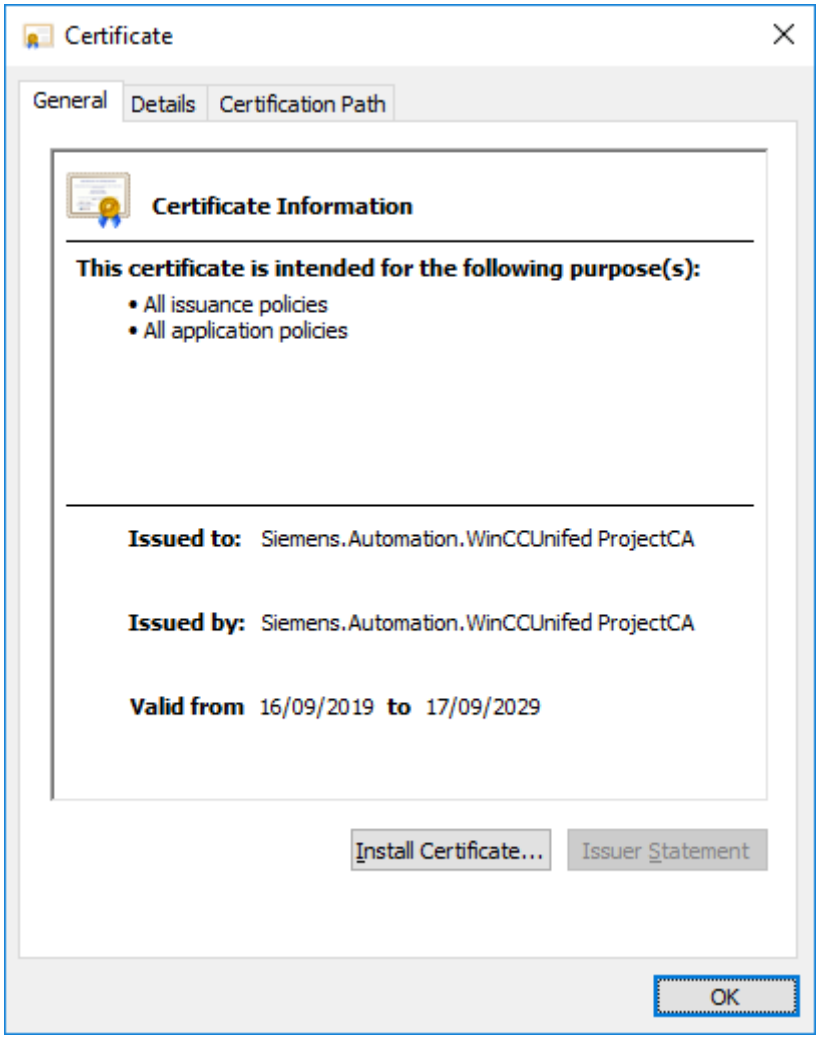


2. Open the field with the error details and confirm that you want to open the web page.
3. On the WinCC Unified home page, select the field "Certificate Authority" and confirm "Open file" in the download dialog.



The root certificate is downloaded to the default download directory.

4. Open the downloaded file.
   The root certificate is opened with the Windows standard form.



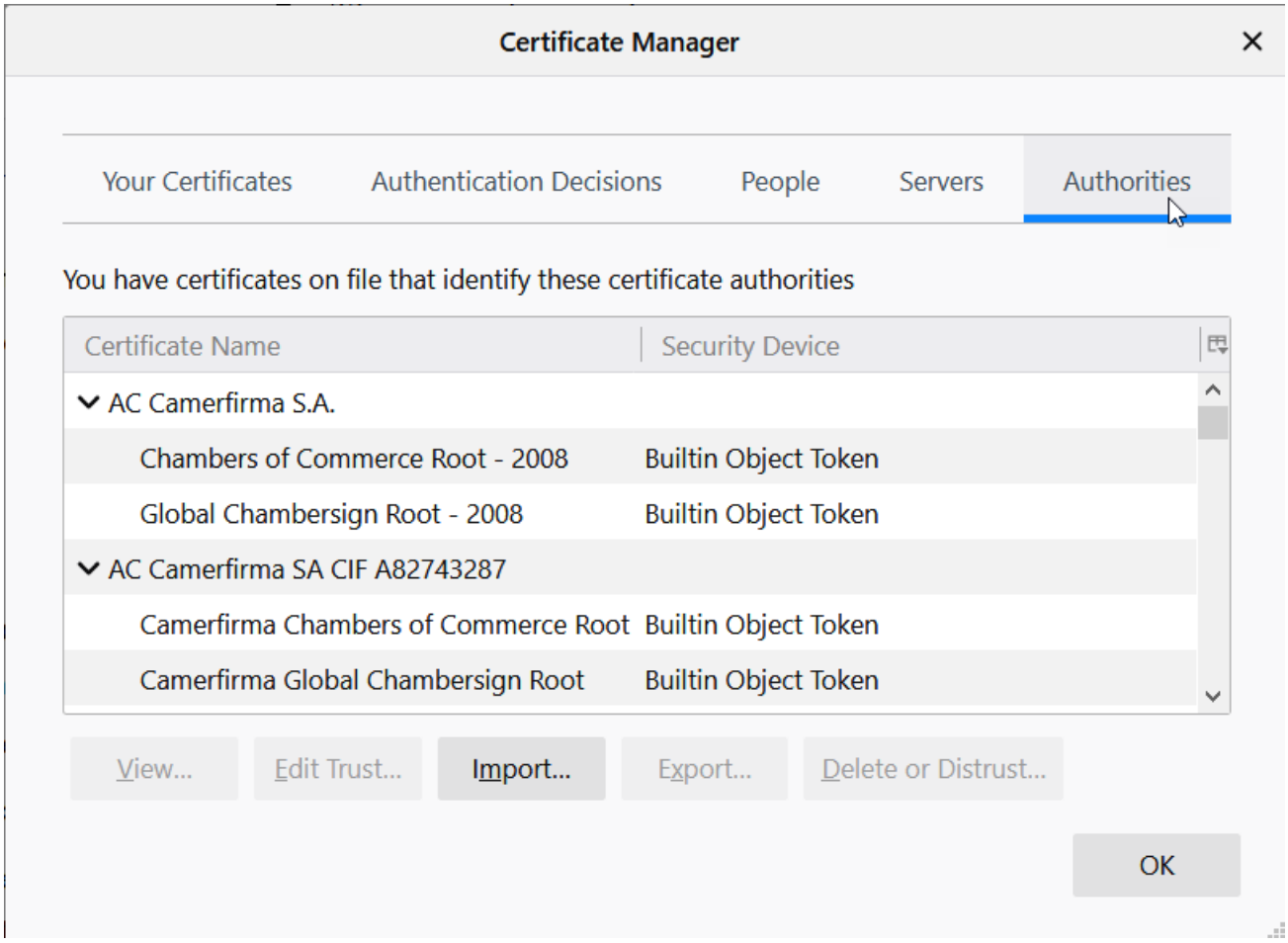
5. To import the root certificate into Windows, select "Install Certificate".
6. In the certificate import wizard, select "Local Machine" as the storage location, "Trusted Root Certification Authority" as the certificate store and start the import process.

Installing the root certificate for Firefox

Firefox uses its own certificate store and must therefore be configured manually on each device once:

1. Open the WinCC Unified home page via the URL https://<host name>
   At first, an error message appears:
2. Open the field "Advanced" and confirm the field "Accept the Risk and Continue".
   An exception is entered for this page in the Firefox certificate management.
3. On the WinCC Unified home page, select the field "Certificate Authority".

4. Save the root certificate. To do this, click "Save file" in the Firefox dialog that follows.
5. Store the certificate in the Firefox certificate store. Proceed as follows:
   - Open the "Settings" page of Firefox.
   - Select "Privacy & Security". There you will find the "Certificates" area further down. Open "Show certificates...".
   - In the "Certificate Management" window, select the "Certification authorities" tab:

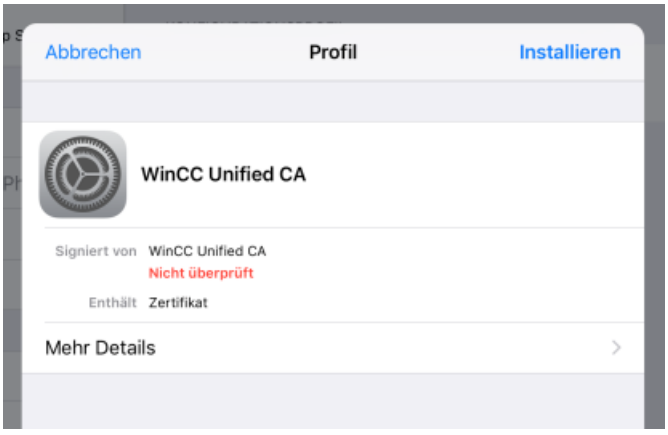


   - Click "Import" and select the root certificate you saved in step 3.
   - In the window that opens, select the option "This certificate can identify websites" and confirm your selection.
   - Click "Server" and remove the exception that was created by step 2.

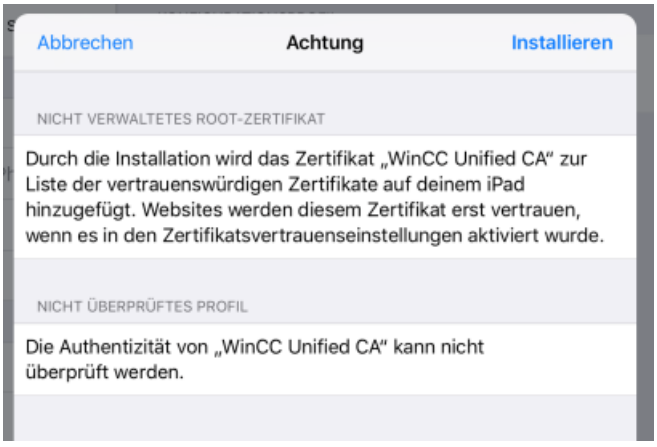
Installing the root certificate on iOS devices

iOS uses its own certificate store and must therefore be configured manually on each device once. An error message also appears when the WinCC Unified home page is opened.

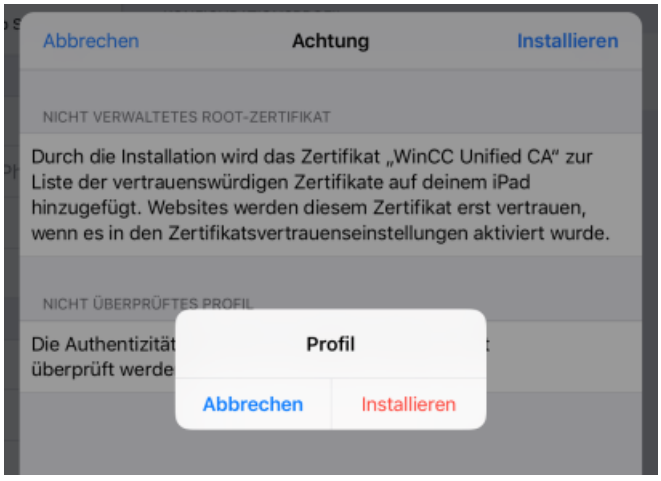
1. Open the field "Advanced" and confirm the field "Accept the Risk and Continue".
2. On the WinCC Unified home page, select the field "Certificate Authority".



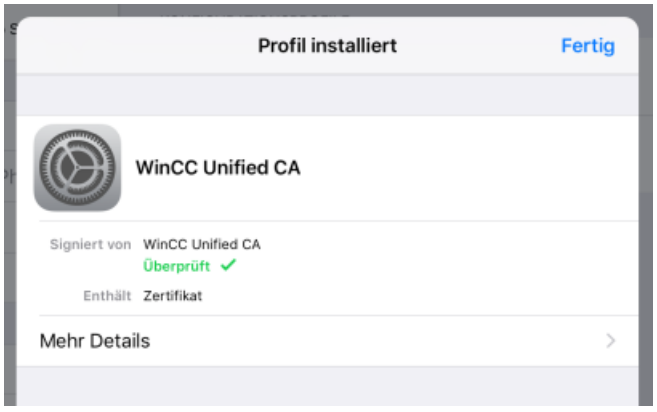
3. Select "Install".



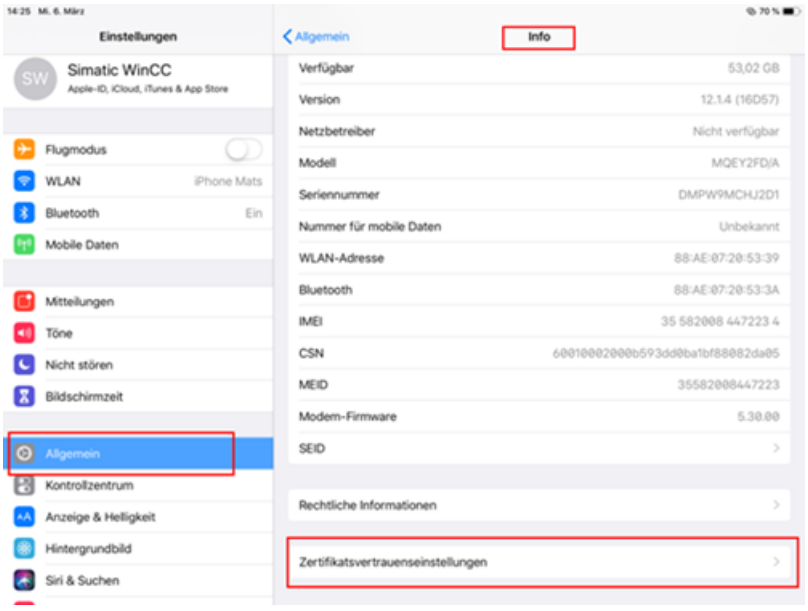
4. Select "Install" again.



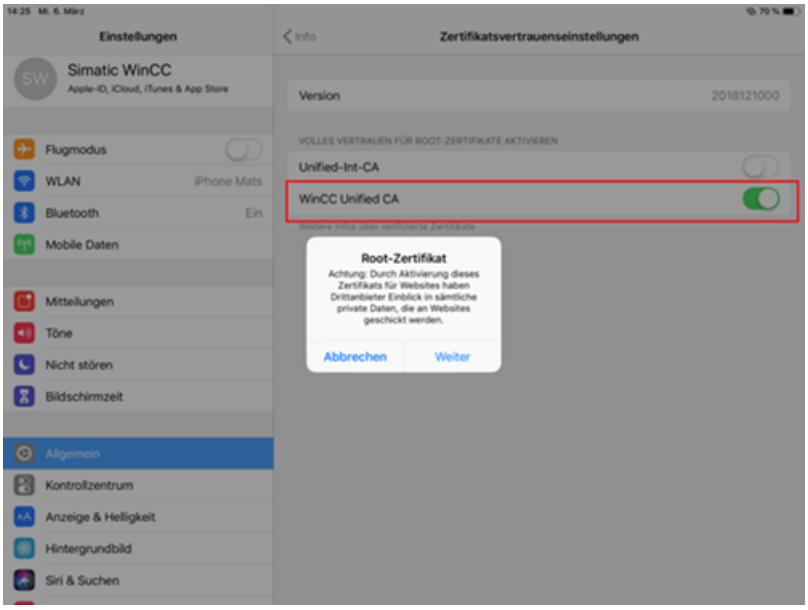
You see the entry "Trusted".



5. Select "General > Info > Certificate Trust Settings".



6. Enable "WinCC Unified CA" and select "Next".



17.5.13 Recreating certificates

WinCC Unified Certificate Manager offers the option to recreate existing certificates.

You must recreate certificates in the following cases:

Expiry of an application certificate	Recreate the application certificate.
Expiry of the root certificate	Recreate the entire configuration of the certificate authority.

Expiry of the CRL file	Update the CRL file.
Change to the IP address or the computer name of a Unified device[1]	Recreate the application certificates of the device.

[1] For a Unified PC that is used as an HMI device as well as a certificate authority device: If you have changed the computer name or the IP address, recreate the entire configuration of the certificate authority. Distribute and install it.
When the certificate authority device is not used as an HMI device, there is no need to renew the certificate configuration.

See also

Recreating application certificates (Page 7553)

Recreating the entire configuration (Page 7552)

Updating a CRL file (Page 7553)

Create backup (Page 7554)

17.5.13.1 Recreating the entire configuration

The expiration of the root certificate requires that the entire configuration of the certificate authority is created again.

Requirement

A certificate authority has been created and configured on the certificate authority device in WinCC Unified Certificate Manager.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
   You will see the configuration of the certificate authority.
3. Right-click the root certificate and select "Recreate all".
4. The "Recreate certificate authority" dialog opens.
   The properties of the previous certificate authority are taken over as default. Change them if necessary.
5. Enter the same password as when you created the certificate authority and confirm it.
6. Click "Create".

Result

The configuration of the certificate authority is recreated:

- Private key
- Root certificate
- CRL file
- All devices and their application certificates

Next steps

- Export the certificate configuration of the devices. Import and install them on the devices.
- Distribute the root certificate and CRL file to the external communication partners.

See also

Exporting root certificate and CRL file (Page 7543)

Exporting, importing and installing for Unified PCs (Page 7536)

Export, import and installation for Unified Comfort Panels (Page 7540)

Creating a certificate authority and root certificate (Page 7530)

17.5.13.2 Recreating application certificates

In the following cases, it is necessary to recreate application certificates:

- The lifetime of a certificate has expired.
- Entries for a valid certificate are to be edited, for example to correct entries.
- The IP address or computer name of the device has been changed.

Procedure

1. On the certificate authority device, open WinCC Unified Certificate Manager.
2. Select the "CA configuration" tab.
3. Right-click on the application certificate of the desired device and select "Recreate".
   The "New <Certificate type> certificate" dialog opens. The entries of the old certificate are downloaded into the dialog.
4. Change the desired properties.
5. Click "Confirm".

Result

A new certificate is created. Export the certificate configuration of his device and install the certificate on the device.

See also

Exporting, importing and installing for Unified PCs (Page 7536)

Export, import and installation for Unified Comfort Panels (Page 7540)

17.5.13.3 Updating a CRL file

When the root certificate is created, the CRL file is given a lifetime of 24 months. When the lifetime expires, it is necessary to update the CRL file.

Requirement

A certificate authority has been created on the certificate authority device in WinCC Unified Certificate Manager.

Procedure

1. On the certificate authority device, open Certificate Manager.
2. Select the "CA configuration" tab.
3. Under the root certificate, click the "Certificate Revocation List" node on the right.
4. Select "Update".

Result

A new CRL file with a lifetime of 24 months is created.

Next step

Distribute and install the CRL file to the target devices:

- On a Unified PC, you install the file with SIMATIC Runtime Manager.
- On a Unified Comfort Panel, install the file in the Control Panel under "Security" using the "Certificates" > "Import" function.
- To distribute files to external communication partners, proceed as described in the user help for the device.

See also

Exporting root certificate and CRL file (Page 7543)

Creating a certificate authority and root certificate (Page 7530)

17.5.14 Create backup

Procedure

To create a backup copy of all the data of the certificate authority, follow these steps:

1. On the certificate authority device, open WinCC Unified Certificate Manager.
2. Select the "CA configuration" tab.
3. Click on the root certificate on the right and select "Export > Full backup".
4. To protect the backup file, enter and repeat a password in the "Export" dialog.
   See also section Password requirements (Page 7521).

5. Click "Export".
6. Click on "Save" and select the storage location and the file name.

Result

The entire configuration of the certificate authority is written to a backup file.

Loading the backup

1. Open Certificate Manager.
2. In the "CA configuration" tab, double-click the "Open configuration ..." entry.
3. Select the backup file and confirm with "Open".
4. Enter the password set when creating the backup and confirm with "Open".

Using distributed systems 18

18.1 Overview

Unified Collaboration

Unified Collaboration gives you the option to monitor and operate the runtime of an HMI device within the runtime of another HMI device.

Both HMI devices must be configured accordingly and located in one network. Unified Collaboration supports cross-project access.

To display the screens of the Collaboration device in runtime, use screen windows.

Web client

By using the web client you can access the runtime of an HMI device from any device.

Each web client is autonomous and independent of the runtime displayed locally on the HMI device. We are distinguishing here between synchronous and asynchronous functions.

SmartServer

With the "SmartServer" option you can access a Unified Comfort Panel using an application.

You activate the SmartServer in the TIA Portal in the runtime settings of the Unified Comfort Panel. You download the application to the device from which you want to access the Panel and execute it.

The user interface of the Panel is mirrored. It will give you access to the entire device including Control Panel and runtime. All actions that you are executing via the application are immediately visible on the Unified Comfort Panel.

18.2 Unified Collaboration

18.2.1 Basics

18.2.1.1 Basics

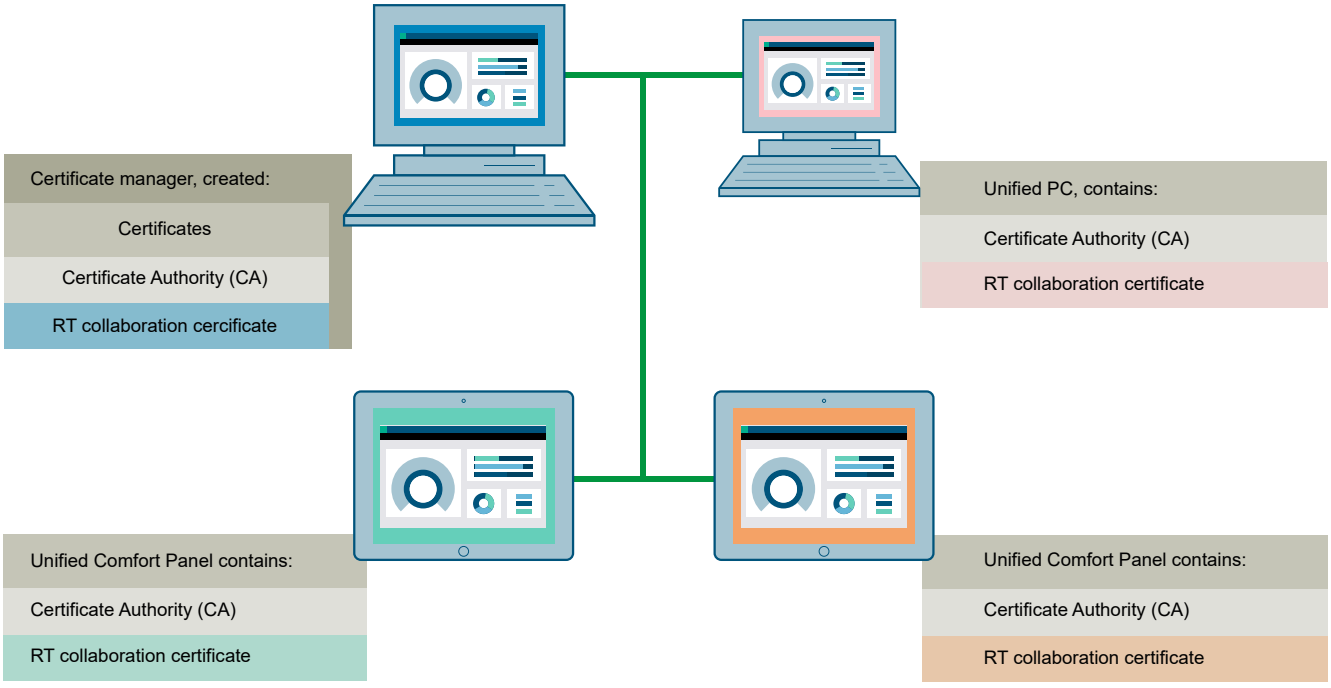
Introduction

You can use Unified Collaboration to access Unified Runtime objects, such as screens of another HMI device. You can display and operate the screens.

You use Unified Collaboration together with:

- SIMATIC WinCC Unified PC
- SIMATIC Unified Comfort Panel

The collaboration devices can be created in the same project or in different projects. The configuration steps are different for both variants.



Unified PC
Unified Comfort Panel
Screen window

Configuration steps for HMI devices in the same project

1. Create a project.
2. Add multiple HMI devices.
3. Create screens for the HMI devices.
4. Making certificates available.
5. Define collaboration settings.
6. Assign the screens to the screen windows.
7. Compile and load all HMI devices.

Configuration steps for HMI devices in different projects

1. Creating multiple projects.
2. Add multiple HMI devices.
3. Create screens for the HMI devices.
4. Making certificates available.
5. Define collaboration settings.
6. Export screen references for Unified Collaboration.
7. Import screen references for Unified Collaboration.
8. Assign the screens to the screen windows.
9. Compile and load all HMI devices.

OPC UA Collaboration

If several HMI devices operate in collaboration as OPC UA servers, the collaboration devices can read and write tags and alarms from the OPC UA servers.

See also

Restrictions (Page 7560)

Creating certificates (Page 7564)

Configuring a screen window within a project (Page 7575)

Defining collaboration settings (Page 7572)

Configuring screen windows from different projects (Page 7577)

18.2.1.2 Requirements

To use Unified Collaboration among several devices, the following requirements must be fulfilled:

- The following software is installed on each participating device:
  SIMATIC WinCC Unified Runtime V18 (basic package, license required)

  Note

  Cross-version use of collaboration

  Screens of an HMI device with installed Runtime version V16 can be displayed in a screen window of an HMI device with installed Runtime version V17 via Unified Collaboration.

  In order that an HMI device with installed Runtime version V16 can participate in collaboration, a collaboration certificate must be created with a Certificate Manager V17 and transferred to the HMI device with installed Runtime version V16. Additional information is available under Using WinCC version compatibility.

- All devices connected via Unified Collaboration must be on the same network and have access to one another.

- Firewall settings of the Unified PC: During TIA Portal setup, the components of the Unified Collaboration are released to the firewall. The security release for the local subnet and can be adapted manually if required. Note the firewall rules "WinCC RTIL dist" and "WinCC RTIL proxy" in this regard.
  The firewall settings are relevant for the collaboration between multiple Unified PCs and for the collaboration between Unified PCs and Unified Comfort Panels.
- Local session tags are supported for V18 or later devices. In a multi-user environment, session-related data is processed independently in each local session. Data from local session tags is not saved after a session is closed.

See also

Introduction to the WinCC Unified Certificate Manager (Page 7517)

Creating certificates (Page 7564)

Using WinCC version compatibility (Page 187)

18.2.1.3 Restrictions

Note the following restrictions when using Unified Collaboration and associated Runtimes:

Screen objects

Unified Collaboration does not support all screen objects. The following screen objects cannot be used in screens that are displayed in another Runtime via Unified Collaboration:

- My controls:
  - Plant overview

System functions and scripts

Executing scripts and system functions depends on the objects used. You can find additional information in the section System functions and scripts (Page 7561).

Subsequent changes to collaboration settings

The following data must not be changed after a connection has been established:

- System ID
- Collaboration name
- IP address / Host name

Synchronicity of the system times

The system times of the interconnected HMI devices must not differ by more than 180 seconds. If the difference is greater, no connection is established between the devices.

Configured languages

The configured and activated languages for Runtime must be the same for all participating devices.

Audit

Audit is not supported in Unified Collaboration.

See also

Defining collaboration settings (Page 7572)

Configuring user management in the engineering system for Runtime (Page 6894)

18.2.1.4 User management

Collaboration supports the cross-project use of multiple users.

Note

Central user management (UMC)

If you want to use Unified Collaboration across projects, make use of central user management and the following advantages:

- Low configuration effort
- Organization of users in user groups
- High security through consistency

You can restrict access to screen objects by using function rights. You configure and manage function rights in the user management.

When you limit the operation of a screen object with a function right and make the screen object available in another HMI device using Unified Collaboration, the user logged into Runtime must have the function right through the assigned roles.

The name of the function right at the screen object is compared to the name of a function right that can be assigned to the logged in user to see whether they match. Access is granted as configured when there is an agreement. It does not matter in this case whether Unified Collaboration is used in one project or across projects.

To avoid confusion, use the same roles and function rights within a system.

18.2.1.5 System functions and scripts

Introduction

You use system functions and scripts in Unified Collaboration in the usual way. Depending on the object, there are some restrictions and special features.

Executing system functions and scripts depends on the objects used.

The objects are arranged in four categories:

- Objects of the Runtime system with system name prefix
- Objects of the Runtime system without system name prefix
- Objects related to the current session
- System functions under "HMIRuntime.Device.SysFct"

"Screen name" parameter

In system functions and scripts, you specify the "Screen name" parameter of a collaboration device in the format "Collaboration Name :: Screen name", e.g. "HMI_RT_1 :: Screen_1".

"SetPropertyValue" system function

The "SetProperty Value" system function, for example, can be used to change the "Screen" property of a screen window. For the "Value" parameter, you can specify the screen of a collaboration device in the format "Collaboration Name :: Screen name".

Note

The collaboration name and the associated screen name are not referenced in the "SetPropertyValue" system function. If you change the collaboration name or the screen name, you must adapt the system function.

Objects of the Runtime system with system name prefix

Scripts and system functions that use objects with a system name prefix are executed on the local HMI device. The objects are read from the collaboration device or written to the collaboration device.

The following objects have a system name prefix, for example:

- "Alarm"
- "AlarmLogging"
- "Tag"
- "TagLogging"
- "LoggedTag"
- "Connection"
- "PlantObject"
- "Report"
- "TextList"
- "Screen" and "ScreenItem"
- "FaceplateInterface"
- "ParameterSetTypes"

Example: Objects of the Runtime system with system name prefix

You configure two HMI devices and activate Collaboration:

- "HMI_1" with a "Screen_1" screen
- "HMI_2" with a "Screen_Collaboration" screen and "Tag_1" tag

Configure a button in the "Screen_Collaboration" screen. Configure the "SetTagValue" system function to an event of the button. Enter the following parameters:

- Tag: "Tag_1"
- Value: "1"

In the "Screen_1" screen, configure a screen window in which the "Screen_Collaboration" screen is displayed.

If Runtime is operating and the event is triggered, the system function is executed on the "HMI_1" HMI device. The value of the "Tag_1" tag changes on the "HMI_2" HMI device.

Objects of the Runtime system without system name prefix

No collaboration device can be addressed for objects without a system name prefix. The relevant system functions and scripts are executed on the local HMI device. The objects are also obtained from the local HMI device. The following objects have no system name prefix:

- "Database"
- "FileSystem"
- "Timers"
- "Traces"

Objects related to the current session

System functions and scripts that contain objects related to the current session are executed on the local HMI device. When executing the scripts and system functions, the effects will show up in the current session of the user currently logged on.

The following objects refer to the current session, for example:

- "DataSet"
- "Language"

The following system functions relate to the current session:

- "Logoff"

System functions under "HMIRuntime.Device.SysFct"

System functions under "HMIRuntime.Device" are disabled in Unified Collaboration. The following system functions cannot be used:

- "CreateScreenshot"
- "EjectStorageMedium"
- "GetBrightness"

- "GetDHCPState"
- "GetIPV4Address"
- "GetNetworkInterfaceState"
- "GetSmartServerState"
- "SetBrightness"
- "SetDHCPState"
- "SetIPV4Address"
- "SetNetworkInterfaceState"
- "SetSmartServerState"
- "ShowControlPanel"
- "ShowSoftwareVersion"
- "StartProgram"
- "StopRuntime"

See also

Restrictions (Page 7560)

18.2.2 Preparing Unified Collaboration

Below you will find out which steps are required to use Unified Collaboration. You create and distribute certificates, configure system alarms and specify the settings for collaboration.

This section includes information on the following topics:

- Creating certificates (Page 7564)
- Distributing and installing certificates (Page 7567)
- Configuring system events for Unified Collaboration (Page 7570)
- Defining collaboration settings (Page 7572)
- Changing the collaboration settings (Page 7574)

For more information, refer to the online documentation for the "Certificate Manager".

18.2.2.1 Creating certificates

Introduction

To enable collaboration between different collaboration devices, the corresponding certificates must be provided in advance. All certificates are issued by a common Certification Authority (CA) to simplify the trust relationship between the communication partners.

The root certificate of the CA is classified as trustworthy on each device for each application. When accessing WinCC Unified Runtime via websites, the root certificate must be configured as trustworthy once in the web browser.

To enable collaboration, you need a "Runtime Collaboration certificate" for each collaboration device in addition to the root certificate.

You create and manage the certificates with the WinCC Unified Certificate Manager.

Note

Renewal of the Runtime Collaboration certificate after upgrading to V18

After a Runtime Collaboration device has been upgraded to V18, its Runtime Collaboration certificate must be renewed.

Additional information is available under Using WinCC version compatibility (Page 187).

Note

Cross-version use of collaboration

Screens of an HMI device with installed Runtime version V17 can be displayed in a screen window of an HMI device with installed Runtime version V18 via Unified Collaboration.

To allow an HMI device with installed Runtime version V17 to participate in collaboration, a collaboration certificate must be created with a Certificate Manager V18 and transferred to the HMI device with installed Runtime version V17.

Creating a root certificate

1. Select a WinCC Unified PC device in your network that is to serve as the certification authority. The root certificate and the associated key are only available on this device. The configuration of additional application certificates for other devices is only possible on this device.
2. Open WinCC Unified Certificate Manager on this device.
3. Create a new root certificate. Double-click the "Create new certification authority" button.
4. Enter the properties of the root certificate in the "New certification authority" dialog. The fields are freely editable.
   Mandatory fields:
   - "Certification authority"
   - "Password" for the private key

   If necessary, select a different key length and runtime for the certificate.
5. Click "Create".

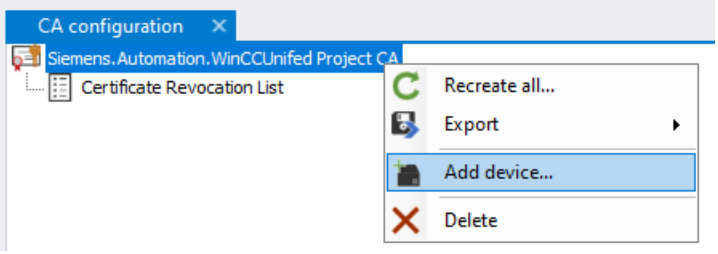
The root certificate and its key are stored on the device and used to generate the device certificates.

Note

If the "WinCC Certificate Manager" is restarted on this device, the root certificate and the device certificates generated with it are loaded automatically.

Adding devices

1. Right-click the root certificate and select "Add device ...".



2. Enter the name and/or IP address of the device in the "New device" dialog box.
   The specification of the IP address is sufficient for Unified Comfort Panel.
   For devices with dynamic IP addresses, enter only the host name.

   Note

   Permitted names

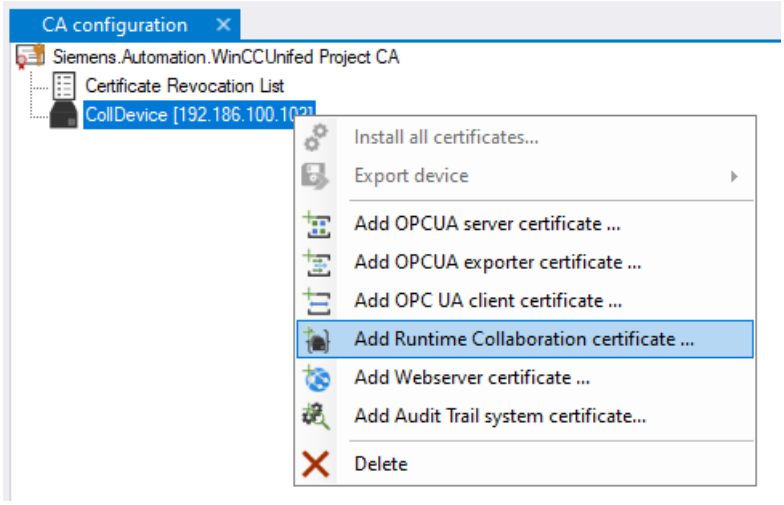
   Either the host name or the "Fully qualified domain name" can be used as the name. The name is inserted in the certificates created for the device and used for validation. To avoid validation errors when accessing the web pages, the "Fully qualified domain name" must be used within a domain.

   Using the name "localhost" is not permitted and is automatically replaced with the name of the local device by the Certificate Manager.

3. Repeat steps 1 and 2 until all collaboration devices have been added.

Adding a "Runtime Collaboration certificate"

1. Right-click on a device and select "Add Runtime Collaboration certificate ...".



2. Enter the properties of the certificate in the dialog.
   If necessary, select a different key length and runtime for the certificate.
3. Create a "Runtime Collaboration certificate" for each collaboration device.
4. If necessary, create additional certificates, e.g. web server certificates.
   You can find additional information in the "WinCC Unified Certificate Manager" operating manual.

Result

- You have created the root certificate.
- You have added all collaboration devices.
- You have added a "Runtime Collaboration certificate" for all collaboration devices.
- You have created additional certificates as required.

See also

Basics (Page 7557)

Requirements (Page 7559)

Using WinCC version compatibility (Page 187)

18.2.2.2 Distributing and installing certificates

Introduction

To distribute the configured certificates to the corresponding devices, the certificates must be exported to a secure storage file. This file must be transferred manually to the respective device and imported there. The procedure is different for Unified PC and Unified Comfort.

Trust communication partners

For successful communication, the collaboration device must trust the root certificates of its communication partners and vice versa.

Trust relationship between collaboration devices

Collaboration devices whose certificate configuration comes from the same certificate authority automatically trust each other after the certificate configuration is installed.

Trust relationship with external communication partners

To establish the trust relationship with an external communication partner, follow these steps:

1. On the certificate authority device or on the collaboration device:
   Export the root certificate and CRL file to an external data storage medium.
2. On the external communication partners:
   - Connect the external communication partner to the external data storage medium.
   - Copy the files and trust them. To do this, proceed as described in the user help of the device.
   - Export the root certificate of the external communication partner and its CRL file to the external data storage medium.
3. Connect the external data storage medium to the collaboration device.
4. Import the root certificate and the CRL file of the external communication partner into the collaboration device and trust it:
   - Unified PC: Use SIMATIC Runtime Manager.
   - Unified Comfort Panel: In the Control Panel of the device, under "Security", use the "Certificates" > "Import" function.

During the next connection attempt, the communication partners mutually accept their application certificates.

Exporting the certificate configuration

After you have completed the certificate configuration of the devices, export the certificate configuration.

The procedures differ for PC devices and Panel devices.

Note

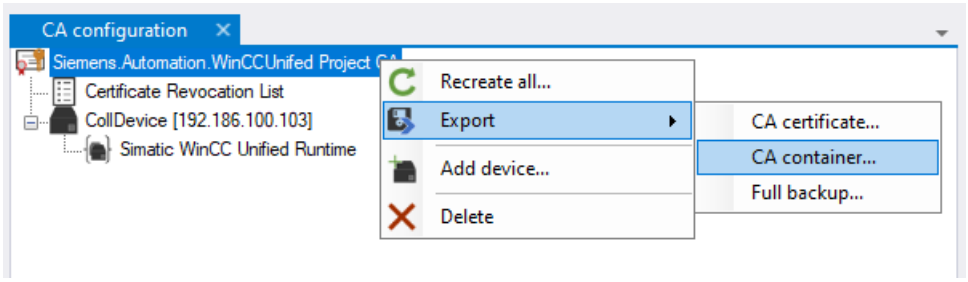
Distribution of root certificate and CRL file

Root certificate and CRL file are part of the certificate configuration of a device. They are exported or imported with the certificate configuration, installed and classified as trusted.

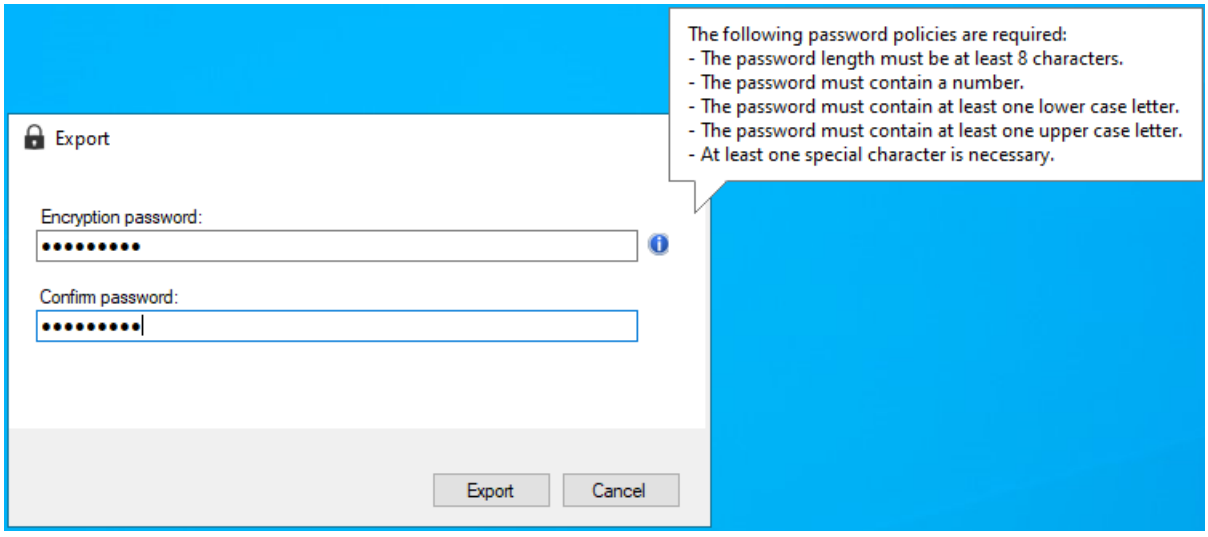
On Unified PCs, you can also export the root certificate and the CRL file via the SIMATIC Runtime Manager.

Exporting all certificates for Unified PC

1. Right-click the root certificate and select "Export..." > "CA container" in the menu.



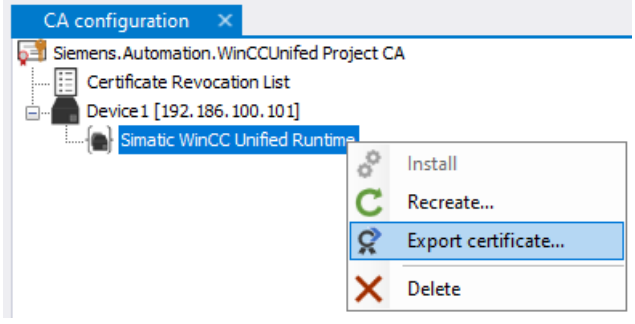
2. Assign a password in the "Export all" dialog.



3. Click on "Export" and select the storage location and file name.
   The data is stored encrypted with the specified password.

Exporting the certificate for a device

1. Right-click on the certificate and select "Export certificate" in the menu.



2. Select the format for the exported certificate.
3. Select the storage location.

Making certificates available on the Unified devices

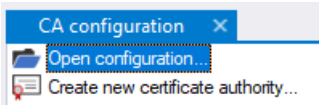
The procedures differ for PC devices and Panel devices.

Depending on the device, use the following tool:

- Unified PC: WinCC Unified Certificate Manager
- Unified Comfort Panel: Control Panel > "Security" function

To make the certificates of a Unified HMI device available on the Unified PC, follow these steps:

1. Transfer the CA container or the certificates to the Unified PC.
2. Open the Certificate Manager.
3. Import the certificate configuration.



To install an individual certificate of the device:

1. Under the local machine node, right-click the certificate.
2. Select "Install".

Result

You have distributed and installed the required certificates on all collaboration devices.

The Runtime Collaboration certificate only becomes effective after a restart of the WinCC Unified Runtime.

18.2.2.3 Configuring system events for Unified Collaboration

Introduction

An alarm log must be configured in advance in order to display system events relating to collaboration in the alarm view of a collaboration device. The alarm log supports you in identifying connection problems, for example, when a connection cannot be established due to an incorrect system ID.

Note

System events for collaboration

System events for collaboration are only available for the current HMI device and the HMI devices that you select in the collaboration settings under "Connect actively to".

Requirement

- A Unified device has been configured.

Procedure

1. Open the "Logs" editor in the project tree.
2. Switch to the "Alarm logs" tab.
3. Create an alarm log.
4. Make the required settings for the alarm log.
5. Open the "HMI alarms" editor in the project tree.
6. For the alarm class "SystemNotification", define the alarm log created in the "Log" column.
7. If required, you can define the alarm log for other classes.

Result

System events for collaboration are displayed in Runtime as logged alarms in the alarm view of the collaboration device.

The system events contain, for example, information on the connection status. Alarms for each connected device are displayed.

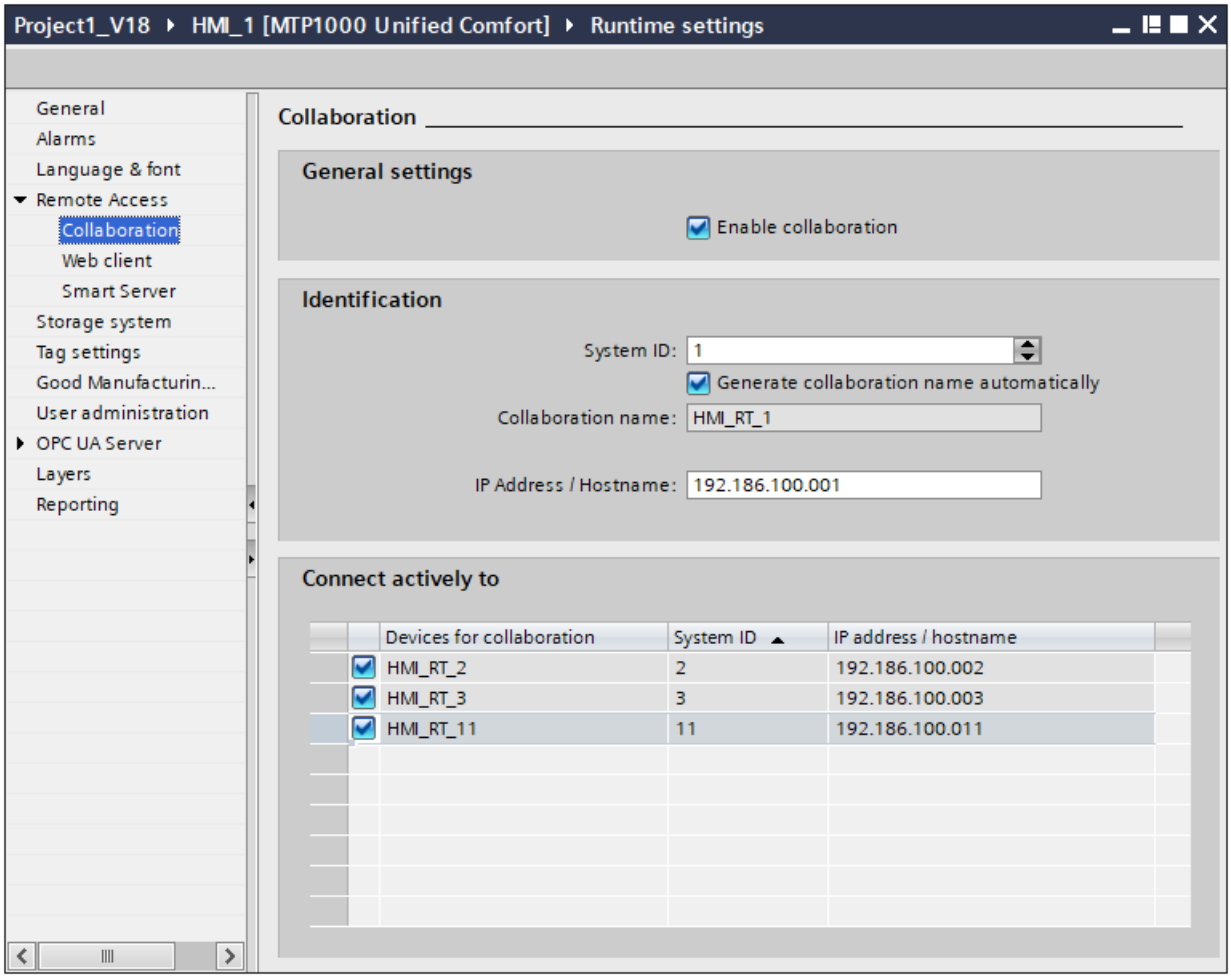
See also

Log basics (Page 837)

Logging alarms (Page 769)

18.2.2.4 Defining collaboration settings

A device must be uniquely identifiable in order for it to participate in Unified Collaboration. You make the settings in the "Collaboration" or "Remote access > Collaboration" area of the Runtime settings of the HMI device.



Identification

Note

The following information must be different for all devices participating in Unified Collaboration:

- System ID
- Collaboration name
- IP address / Host name

If a collaboration device cannot be uniquely identified in all of its information, the configuration is invalid. Errors can occur during the further configuration.

- System ID
  This ID is used for communication between the devices.
  The value can be between 1 and 2046. The system ID must be unique for devices within the project.
- Collaboration name
  If "Generate collaboration name automatically" is enabled, the collaboration name corresponds to the device name. Changes to the device name are transferred automatically.
  If "Generate collaboration name automatically" is disabled, assign the collaboration name manually. The assigned name must be unique across the system and can consist of up to 128 characters.
- IP address / Host name
  The IP address must correspond to the IPv4 format. All devices connected via Unified Collaboration must be in the same network. The IP address or the host name must be unique.

Connect actively to

Lists all HMI devices available for collaboration and shows the system ID and IP address/host name.

To select a device as collaboration device, activate the device.

Note

System events for collaboration

System events for collaboration are only available for the current HMI device and the HMI devices that you select under "Connect actively".

The system events contain, for example, information on the connection status. Alarms from each connected device are displayed. System events are displayed in the "Alarm view" control as logged alarms.

Requirement

- An HMI device has been created.

Procedure

To define the settings for Unified Collaboration, follow these steps:

1. Open the "Devices" tab in the project tree.
2. Open the Runtime settings of the respective HMI device.
3. Switch to the "Collaboration" area.
   With Unified Comfort Panel you will find the "Collaboration" area under "Remote access".
4. Select the check box "Enable collaboration" under "General settings".

5. Assign a system ID for the HMI device under "Identification".

   Note

   The system ID is incremented separately for different projects. To ensure a unique assignment, change the ID manually.

6. Specify the collaboration name.
7. Enter the IP address or the hostname of the device.
8. Select the collaboration devices under "Connect actively to".
   In order for a device to be visible under "Connect actively to", the "Enable collaboration" option must be activated for this device.

See also

Restrictions (Page 7560)

Basics (Page 7557)

Changing the collaboration settings (Page 7574)

18.2.2.5 Changing the collaboration settings

Introduction

A device must be identifiable by a unique system ID for it to participate in Unified Collaboration. When you make changes in the Runtime settings of the HMI device under "Collaboration", you must ensure that the information is valid for all devices participating in Unified Collaboration.

Note

When you change the collaboration settings of a device participating in Unified Collaboration, you must completely compile and download the projects of all participating devices.

Changing the collaboration settings

To change the collaboration settings, follow these steps:

1. Change the collaboration settings of an HMI device.
2. Check whether the following information is different for all devices participating in Unified Collaboration:
   - System ID
   - Collaboration name
   - IP address / Host name

   Invalid entries are highlighted in red.

3. If all devices participating in Unified Collaboration are in one project, follow these steps:
   – Perform a full compilation and download of the projects of all participating HMI devices.
4. If the screen references of the device whose collaboration settings were changed are made available in other projects, follow these steps:
   – Export the screen references of the HMI device.
   – Import the screen references of the HMI device into another project.
   The existing data is overwritten.
   – Perform a full compilation and download of the projects of all participating HMI devices.

See also

Defining collaboration settings (Page 7572)
Complete reloading of a project (Page 7187)
Basics of downloading projects (Page 7182)
Basics for downloading projects (Page 7147)
Complete reloading of a project (Page 7152)

18.2.3 Using Unified Collaboration

18.2.3.1 Configuring a screen window within a project

Requirement

- The collaboration devices are in the same network and have access to one another.
- The required certificates are provided.
- The Runtime settings of the collaboration devices are configured.
- Screens are configured for the collaboration devices.

Procedure

1. Open the screen on the device on which you want to display screens of the collaboration device.
2. Add a screen window to the screen.
3. Open the Inspector window under "Properties > Properties > General > Screen".
4. Click the selection button in the "Static value" column.
   A new dialog opens.
5. Select the collaboration device in the left area of the window.

6. Select the screen to be displayed in the right area of the window.
7. Confirm your selection.

Note

When you delete a device that is participating in collaboration, the links of its screens to the screen windows of other HMI devices are removed as well.

When the same device is to participate in collaboration again, the screens must be linked once again to the screen windows.

Using scripts

Alternatively, you can also configure screen windows via a script:

1. Perform steps 1 and 2 as described above.
2. Open the Inspector window.
3. Configure a script:
   - For an event
   - For an object property
4. Enter the following line:
   `Screen.Windows("Screen window_1").Screen = "HMI_RT_1::Screen_1";`
   Alternatively, use the snippet "Change Screen in Screen window of current screen".
5. Adapt the name of the screen window, the HMI device and the screen.
   In the example, "Screen window_1" is the screen window of the current device and "HMI_RT_1::Screen_1" is the screen of the device connected via Unified Collaboration.

Result

After compiling and downloading, the screen of the collaboration device is displayed in screen window in Runtime.

See also

Basics (Page 7557)

Restrictions (Page 7560)

18.2.3.2 Configuring screen windows from different projects

Export screen references for Unified Collaboration

Introduction

To use Unified Collaboration, the screen references of the screens that are to be available in the Runtime of another HMI device must be exported.

Once this file has been imported into a project, you can use the screens for Unified Collaboration.

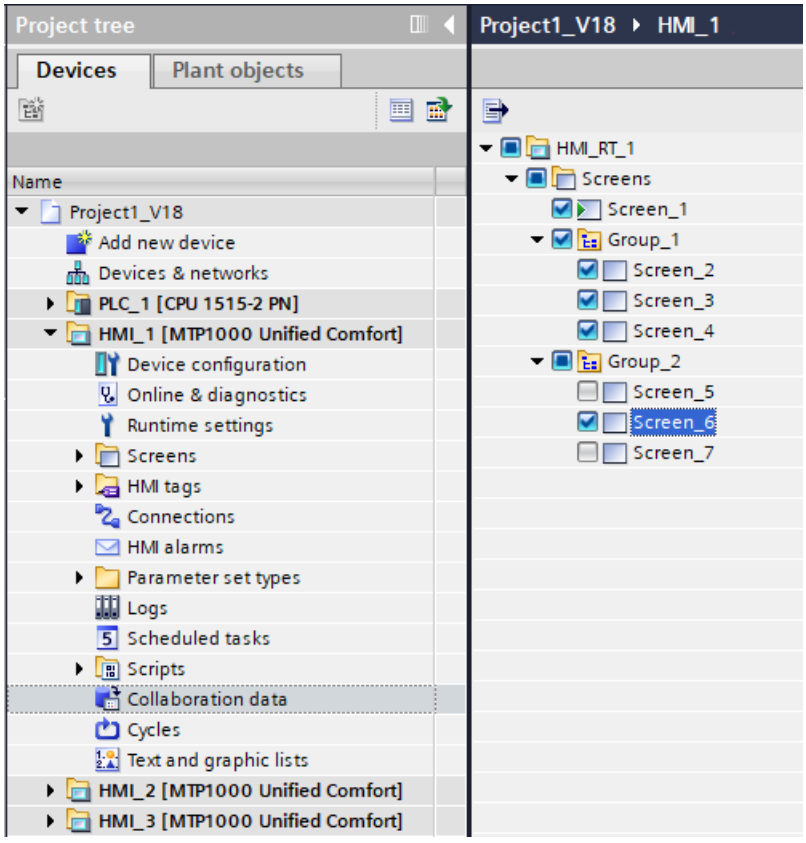
Requirement

- In the Runtime settings, the IP address or host name of the HMI device that provides the screens for collaboration is specified under "Collaboration".
- At least one screen is configured.
- The required certificates are provided.

Procedure

1. Expand the folder of the device in the "Devices" tab of the project navigation.
2. Open the "Collaboration data" editor.

3. Expand the device.
   All available screens of the device are listed.



4. Select the screens you want to export. To do this, put a check mark in the "Export" column of the screen or the group.
5. Click the export icon .

Note

If collaboration is disabled in the Runtime settings of the HMI device, the "Unified Collaboration Export" dialog appears. The export can only be continued when the entries in the Runtime settings are complete:

- System ID
- Collaboration name
- IP address / Host name

A new window opens.

6. Enter a name under which the file is to be saved.
   The "Export completed" dialog box appears.

Note

Only screen references from a single device can be exported with each export operation. To export the screen references of another device, switch to the device and repeat the operation.

Result

You have exported the screen references of the device. You can import the screens into any project.

Note

The exported xml file must not be modified manually.

Import screen references for Unified Collaboration

Introduction

To use Unified Collaboration, import screen references from collaboration devices. These devices can be either HMI devices of the same project or of another project.

Requirement

- The required certificates are provided.
- The screen references of a collaboration device are exported.

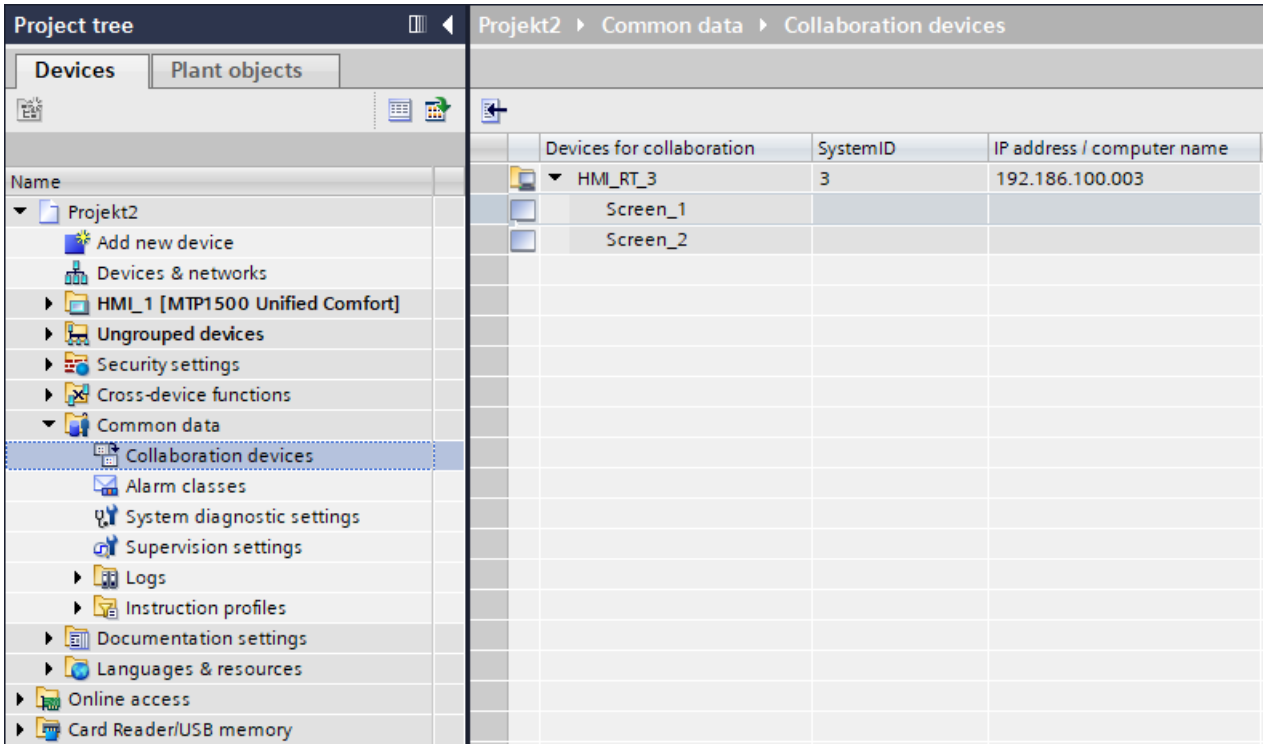
Note

The exported XML file must not be modified manually to ensure error-free import and use of the screens in Runtime.

Procedure for initial import

1. In the "Devices" tab of the project navigation, expand the "Common data" folder > "Collaboration devices".
2. Open the "Collaboration Devices" editor.
3. Click the import icon  .
   A new window opens.
4. Select the xml file you want to import.

5. Confirm all dialogs.
   The "Import completed" dialog box appears.
   The HMI device whose screen references you have imported now appears in the list.



6. Expand the list of the newly imported HMI device to see the screens.

Note

Only screen references from a single device can be imported with each import process. If you want to import the screen references of another device, repeat the procedure.

Procedure for repeated import

If you want to once again import screen references that have already been imported, the procedure is the same as for the initial import. The following collaboration settings must not differ from the previously imported device and must be unique across the system:

- Collaboration name
- System ID
- IP address / Host name

If a value is used more than once, the data import cannot be performed. The import is aborted with an error message in this case.

The previous data is overwritten by the successful import.

Deleting a collaboration device

To delete imported Unified devices from the list, click or use the "Delete" command on the shortcut menu. Individual screen references cannot be deleted. Attempting to delete a single screen reference will delete the entire device from the list.

Result

You have made the screens of a collaboration device available for Runtime of an HMI device of the project.

Note

The imported screen references of an HMI are visible to all HMI devices of the current project.

Configuring screen windows from different projects

Requirement

- Screens are configured for the collaboration devices.
- Export and import of the screen references are completed.
- All Runtime settings are configured and the collaboration devices are connected.
- The users in the participating projects are identical.

Note

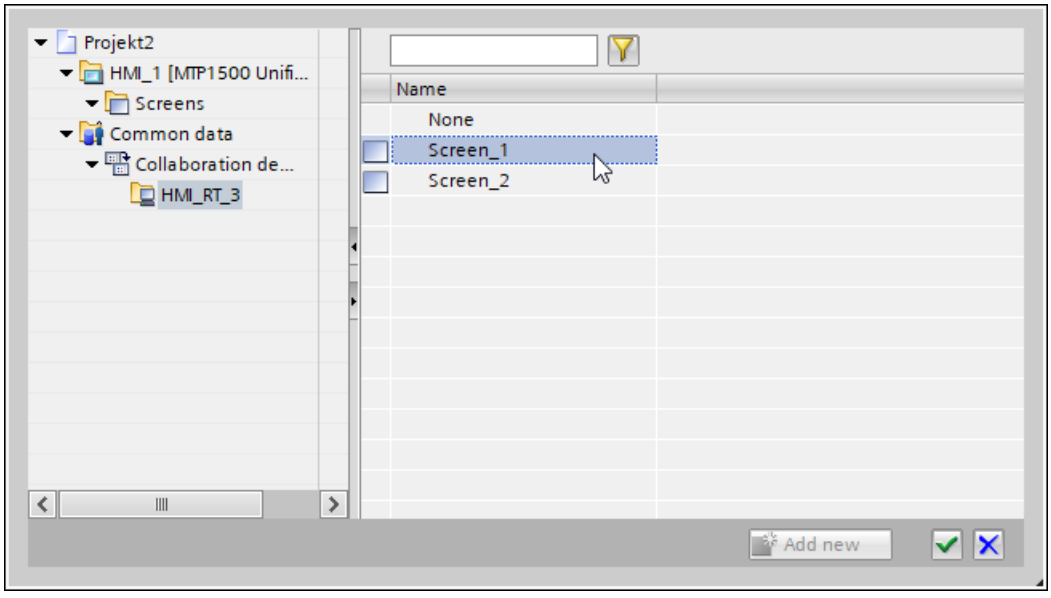
Using the central user administration (UMC)

If you want to use HMI devices in multiple projects for Unified Collaboration, use the central user administration (UMC). This reduces the configuration effort.

Procedure

1. Open the screen on the device on which you want to display screens of the collaboration device.
2. Add a screen window to the screen.
3. Open the Inspector window under "Properties > Properties > General > Screen".
4. Click the selection button in the "Static value" column.
   A new dialog opens.
5. Select the collaboration device in the left area of the window.

6. Select the screen to be displayed in the right area of the window.



7. Confirm your selection by clicking on the green check mark.

Note

When you delete a device that is participating in collaboration, the links of its screens to the screen windows of other HMI devices are removed as well.

When the same device is to participate in collaboration again, the screens must be linked once again to the screen windows.

Using scripts

Alternatively, you can also configure screen windows via a script:

1. Open the screen on the device on which you want to display screens of the collaboration device.
2. Add a screen window to the screen.
3. Open the Inspector window.
4. Configure a script:
   - For an event
   - For an object property
5. Enter the following line:
   `Screen.Windows("Screen window_1").Screen = "HMI_RT_1::Screen_1";`
   Alternatively, use the snippet "Change Screen in Screen window of current screen".
6. Adapt the name of the screen window, the HMI device and the screen.
   In the example, "Screen window_1" is the screen window of the current device and "HMI_RT_1::Screen_1" is the screen of the device connected via Unified Collaboration.

Result

After compiling and downloading, the screen of the connected HMI device is displayed in Runtime in the screen window via Unified Collaboration.

See also

Export screen references for Unified Collaboration (Page 7577)

Restrictions (Page 7560)

18.2.3.3 Display messages from participating devices

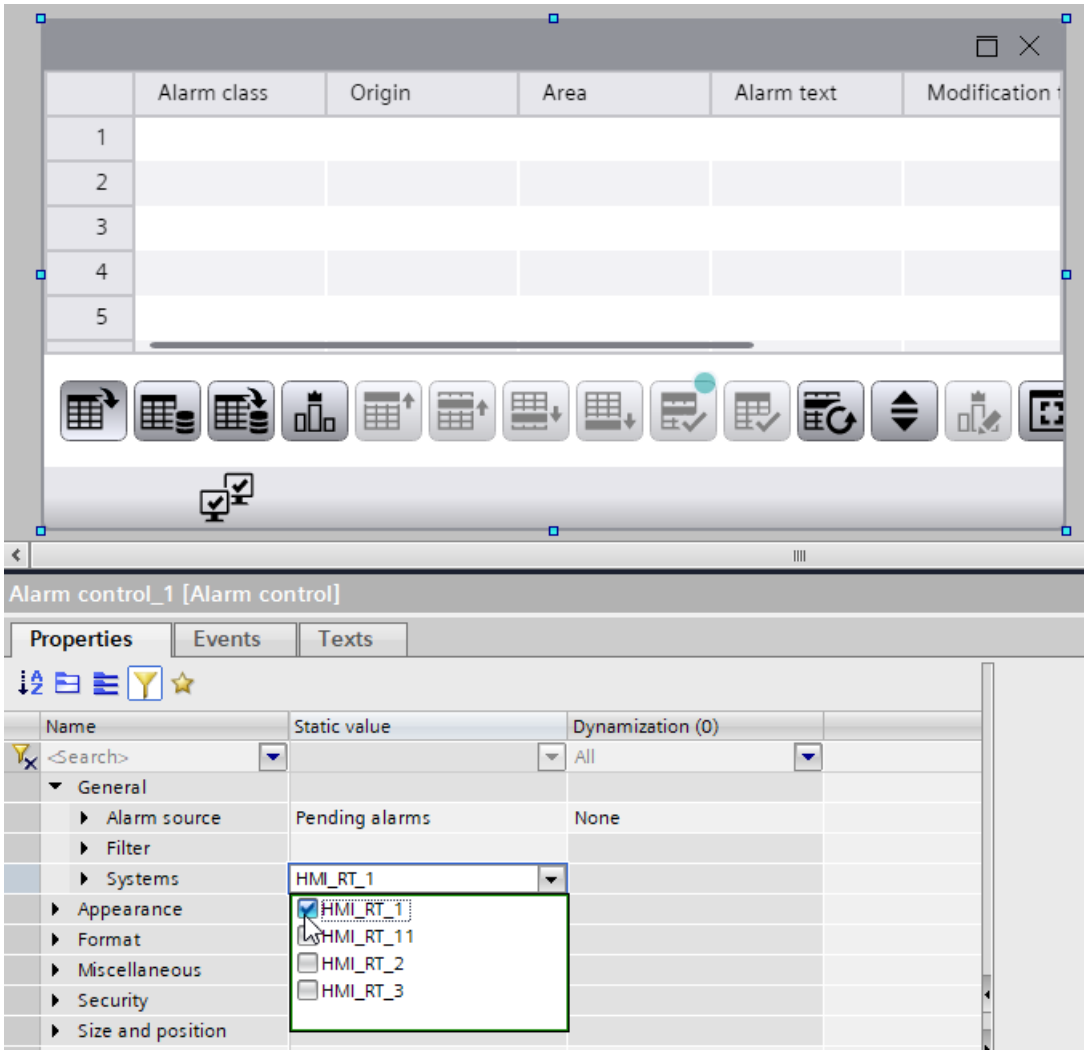
Requirement

Several HMI devices are configured as participating devices for Unified Collaboration.

Procedure

1. Configure a screen for one of the devices.
2. Place an alarm control on the screen.
3. Select the "Systems" property for the alarm control.

4. Expand the list box.
   The participating devices are displayed.
5. Enable the devices whose alarms are to be displayed.



At least one device must be selected.
Default setting: The local device is selected.

Result

In runtime, the alarm control shows the alarms from the selected participating devices.

18.2.4 Example: Connecting HMI devices from two projects with Unified Collaboration

This example shows how to access screen objects of a Unified PC and two Unified Comfort Panels using Unified Collaboration across devices and projects.

Screens of the two Unified Comfort Panels should be operable in the Runtime project of the Unified PC. One of the Unified Comfort Panels is created in another project. The screens of the Unified Comfort Panels are displayed in a screen window. You switch the display of the screens in the screen window by clicking on buttons.

Preparation

- All devices are located in a network and are uniquely identifiable.
- The Unified Collaboration components are enabled in the Windows Firewall.
- The system times of the individual devices differ by less than 3 minutes.

Create and distribute the required certificates using the Certificate Manager:

1. Open the Certificate Manager on the Unified PC.
2. Create a new root certificate for your plant.
3. Create a "Runtime Collaboration certificate" for each device.
4. Transfer the root certificate and one "RT Collaboration certificate" each to the collaboration devices.

Creating projects and adding devices

1. Create two projects: "Project1" and "Project2".
2. Configure a "PC-System_1" Unified PC in "Project1".
   "PC-System_1" is created with the HMI device "HMI_RT_1".
3. Configure a "HMI_2" Unified Comfort Panel in "Project1".
4. Configure a "HMI_3" Unified Comfort Panel in "Project2".
5. Create a "Screen_1" screen for each of the two Unified Comfort Panels.

Runtime settings of the "PC-System_1" Unified PC

1. Expand the folder "PC-System_1" in the "Project tree" under "Devices".
2. Expand the folder "HMI_RT_1".
3. Open the "Runtime settings" of the "HMI_RT_1".
4. Switch to the "Collaboration" area.
5. Activate "Enable collaboration".
6. Check and amend the following settings:
   - The "System ID" is 1.
   - "Generate collaboration name automatically" is selected.
     The "Collaboration name" is "HMI_RT_1".
     This setting cannot be changed.
   - Specify IP address or host name.
     The address must correspond to the IPv4 format.

Runtime settings of the "HMI_2" Unified Comfort Panel

1. Expand the folder "HMI_2" in the "Project tree" under "Devices".
2. Open the "Runtime settings".
3. Switch to the "Collaboration" area.
4. Activate "Enable collaboration".
5. Check and amend the following settings:
   - The "System ID" is 2.
   - "Generate collaboration name automatically" is selected.
     The "Collaboration name" is "HMI_RT_2".
     This setting cannot be changed.
   - Specify IP address or host name.
     The address must correspond to the IPv4 format.

Runtime settings of the "HMI_3" Unified Comfort Panel

1. Open "Project2".
2. Expand the folder "HMI_3" in the "Project tree" under "Devices".
3. Open the "Runtime settings".
4. Switch to the "Collaboration" area.
5. Activate "Enable collaboration".
6. Check and amend the following settings:
   - The "System ID" is 3.
   - "Generate collaboration name automatically" is cleared.
     The "Collaboration name" is "HMI_RT_3".
   - Specify IP address or host name.
     The address must correspond to the IPv4 format.

Exporting screen references

To operate the screens of the Unified Comfort Panels in the Runtime of the Unified PC, the screen references of the Unified Comfort Panel "HMI_3" must be exported from "Project2".

Follow these steps:

1. In the project tree, expand the folder of the respective devices in the "Devices" tab.
2. Open the "Collaboration data" editor.
3. Expand the device by clicking on the arrow.
   All available screens are listed.
4. Select the "Screen_1" screen. To do this, put a check mark in the "Export" column of the corresponding screen.

5. Click Export .
   A new window opens.
6. Enter a name under which the file is to be saved.
   The "Export completed" dialog box appears.

Importing screen references

To operate the screens of the Unified Comfort Panels in Runtime of the Unified PC, the screen references of the Unified Comfort Panel "HMI_RT_3" must be imported into "Project1".

To import screen references, follow the steps below:

1. In the "Devices" tab of the project navigation, expand the "Common data" folder > "Collaboration devices".
2. Open the "Collaboration Devices" editor.
3. Click Import .
   A new window opens.
4. Select the xml file you want to import.
5. Confirm all dialogs.
   The "Import completed" dialog box appears.
   The HMI device whose screen references you have imported now appears in the list.
6. Expand the list of the newly imported HMI device to see the screens.

Connecting actively to collaboration devices

Once the import of the "HMI_RT_3" collaboration device is complete, the devices must be actively connected in the Runtime settings. Follow these steps:

1. Open the Runtime settings of the "HMI_RT_1" Unified PC.
2. Switch to the "Collaboration" area.
3. Activate the collaboration devices "HMI_RT_2" and "HMI_RT_3" under "Connect actively to".
   The collaboration devices "HMI_RT_2" and "HMI_RT_3" are available for collaboration.

Configuring the screen window

In the following, a screen is configured for the Unified PC in which a screen window is inserted. Screens of the two Unified Comfort Panels can be displayed and operated in this screen window.

1. Expand the folder "PC-System_1" in the "Project tree" under "Devices".
2. Expand the folder "HMI_RT_1" and the "Screens" folder.
3. Add a screen.
   The editor of the screen opens.
4. Add a screen window.
5. Select a screen window.
6. Open the Inspector window under "Properties > Properties > General > Screen".

7. Click the selection button in the "Static value" column.
   A new dialog opens.
8. Select "HMI_2 > Screens" in the left area of the window.
9. Select the "Screen_1" screen in the right area of the window.
10. Confirm your selection.

Configuring buttons

To configure the buttons for the screen change in the screen window, follow these steps:

1. Open the screen of the "HMI_RT_1" Unified PC in which the screen window is located.
2. Configure a "HMI_2" button.
3. Configure a "HMI_3" button.
4. Select the "HMI_2" button.
5. Open the Inspector window under "Properties > Events".
6. Select the "Click left mouse button" event.
   The function list is displayed.
7. Select the "ChangeScreen" system function.
8. Open the selection window of the "Screen name" parameter.
9. Select "HMI_2 > Screens" in the left area of the window and confirm the selection.
   The "Screen name" parameter contains the value "HMI_RT_2 :: Screen_1".
10. Specify the type of "Screen window" for the "Screen window path" parameter.
11. Open the selection menu and select the screen window.
12. Select the "HMI_3" button.
13. Open the Inspector window under "Properties > Events".
14. Select the "Click left mouse button" event.
    The function list is displayed.
15. Select the "ChangeScreen" system function.
16. Open the selection window of the "Screen name" parameter.
17. Select "HMI_3 > Screens" in the left area of the window and confirm the selection.
    The "Screen name" parameter contains the value "HMI_RT_3 :: Screen_1".
18. Specify the type of "Screen window" for the "Screen window path" parameter.
19. Open the selection menu and select the screen window.

Starting Runtime

1. Compile and download all HMI devices.
2. Open Unified PC Runtime.

Result

- In Runtime, the screen of the "HMI_2" Unified Comfort Panel is initially displayed in the screen window.
- When you click the "HMI_2" button, the screen of the Unified Comfort Panel "HMI_2" is displayed in the screen window.
- When you click the "HMI_3" button, the screen of the Unified Comfort Panel "HMI_3" is displayed in the screen window.

See also

Defining collaboration settings (Page 7572)

Export screen references for Unified Collaboration (Page 7577)

Import screen references for Unified Collaboration (Page 7579)

Configuring screen windows from different projects (Page 7581)

Creating certificates (Page 7564)

Distributing and installing certificates (Page 7567)

18.3 Web Client

18.3.1 Web client basics

By using the web client you can access the runtime of a Unified PC or Unified Comfort Panel in the network from any device. The accessing device can be a PC or a smartphone, for example.

Access to the web client

By default, you use the web client for local access to your Unified PC runtime.

For Unified Comfort Panel, you must activate the web client in the TIA Portal in the Runtime settings of the Unified Comfort Panel. Alternatively, you can manage the web client directly in the Control Panel of the Unified Comfort Panel.

When you access a Unified PC or a Unified Comfort Panel for the first time, you load a certificate to enable a secure connection. Afterwards, you operate the runtime as always.

Each web client is autonomous and independent of the runtime displayed locally on the HMI device. We are distinguishing here between synchronous and asynchronous functions.

You can access a runtime simultaneously from multiple web clients.

Note

Use the latest web browser version

Use the latest version of your web browser to access the web client.

License

For remote access to a WinCC Unified Runtime with web client, you need a license depending on the HMI device used, the number and the type of accesses.

See also

Opening local user management in the "Browser" screen object (Page 6928)

18.3.2 Mode of operation of the web client

Each web client is autonomous and partially independent of other web clients and the runtime displayed locally on the HMI device. For the runtime to run without errors and expediently, we distinguish between asynchronous and synchronous functions. Some functions are not supported at all or only partially supported in the web client.

All languages configured for the runtime project are supported in the web client.

Synchronous functions

When synchronous functions are executed, the change is applied to the local runtime and in the runtime of other web clients.

Synchronous functions are all changes that change process data and project-related properties, such as:

- Change tag values
- Acknowledging alarms
- Changing log entries

Note

Changes of properties at screen objects

Properties of screen objects that were changed through a user action are not applied to the local Runtime and other web clients.

Example: Color, height and width of screen objects

Asynchronous functions

Asynchronous functions are executed independent of the local runtime and other web clients.

Asynchronous functions are mainly customizable display settings, such as:

- Change screen
- Change language
- Zoom

- Filter
- Display in screen windows

Scripts and system functions

Scripts and system functions are mainly executed on the HMI device on which the runtime is running.

A few system functions are executed on the device on which the web client is run, such as:

- "ExportParameterSets"
- "ImportParameterSets"

System functions under "HMIRuntime.Device.SysFct"

The following system functions under "HMIRuntime.Device.SysFct" are partially supported:

- "SetDHCPState":
  The "mode" parameter can be changed.
- "SetNetworkInterfaceState":
  The "adapterName" parameter can be changed.

The following device-dependent system functions under "HMIRuntime.Device.SysFct" are executed on the HMI device that is accessed via the web client. The effect of these system functions is not visible via the web client.

- "EjectStorageMedium"
- "StopRuntime"
- "ShowControlPanel"
- "ShowSoftwareVersion"
- "StartProgram"

Web client on the Unified Comfort Panel

If you want to access another HMI device from the Unified Comfort Panel using the web client, use the "Browser" control.

Option to changer user

If you use runtime locally on the Unified Comfort Panel, you will be shown an option to switch the user if you do not have sufficient functional rights. This option does not apply to access via a web client. In this case, a note about insufficient authorizations is displayed in the web client.

18.3.3 Activate web client for Unified Comfort Panel

Requirement

- A project has been created.
- A Unified Comfort Panel has been configured.

Enabling the web client in the TIA Portal

1. Open the runtime settings of the Unified Comfort Panel.
2. Expand the "Remote access".
3. Select "Web Client".
4. Enable the "Enable web access to runtime" option.
   A note about licensing is displayed.

Enabling the web client in the Control Panel

1. Open the Control Panel of the Unified Comfort Panel.
2. Select "Runtime properties > Web client".
3. Select "Enable web access to runtime".

18.3.4 Using the web client

Requirement

- The web client is enabled when a Unified Comfort Panel is used.
- The project has been compiled without errors and loaded.
- The project runs in runtime.

Procedure

To access the runtime of a different HMI device using the web client, follow these steps:

1. In the web browser, specify the IP address or the fully qualified name (name and domain) of the HMI device running runtime.
   If you want to access the runtime of a Unified Comfort Panel, use the IP address, for example "https://141.73.65.245/".
   If you are using a browser that runs directly on the HMI device, you can also use "localhost" instead of the IP address.
   To access the Runtime project directly, use "https://<ip>/device/WebRH", note the use of lowercase/uppercase letters, and use the IP address of the HMI device instead of the placeholder "<ip>".
2. If you access the runtime of the HMI device from this device for the first time and there is no corresponding certificate, you will receive a security warning. Follow these steps:
   - Continue loading the web page.
   - Select "Certificate Authority" or the symbol or the message "Not secure" in the address line of the browser.
     The certificate is downloaded.
   - Install the certificate in the web browser.
   - Reload the page.

   Note

   If you reset the time on the web client after installing the certificate so that it is before the time the certificate was created, the certificate is invalid. Install a valid certificate.

   The WinCC Unified home page is displayed.
   After complete download of a project, an error can occur when you open the WinCC Unified home page (SwacLogin).
   You can find additional information at SwacLogin: Errors after complete download (Page 7601).

   Note

   If you experience display problems in the web client, completely delete the browser data (history, form entries, etc.).

3. Select "WinCC Unified RT".
   The user login appears.
4. Enter the user name and password and select "Log in".

Result

The runtime of the remote HMI device is displayed.

The Runtime project is displayed in the language that is set in the "User login" dialog. If this language is not configured in the Runtime settings of the HMI device, the language that has the lowest number in the "Order" column under "Runtime settings > Language & font" in the TIA Portal is used.

Depending on the access authorization, you the option to monitor and operate the runtime.

See also

Installing a certificate in the browser when accessing via web client (Unified PC) (Page 7594)
Installing a certificate in the browser when accessing via web client (UCP) (Page 7601)

18.3.5 Installing a certificate in the browser when accessing via web client (Unified PC)

Using root certificates

To enable web browsers to establish a secure connection to WinCC Unified, the current root certificate of the WinCC Runtime must be known in the web browser as a trusted certification authority.

By installing the web server certificate on the PC device, the public root certificate is made available as a download for installation in web browsers on the WinCC Unified home page.

The procedure for installing the root certificate differs depending on your web browser.

Use of self-signed certificates

As an alternative to the root certificate, you can use a self-signed certificate.

NOTICE
Security risk from self-signed certificate A self-signed certificate is not issued by a trusted certification authority. If you use a self-signed certificate from an untrustworthy source, the data transfer is not protected from attacks. Before using self-signed certificates, check the source. Depending on the firewall and network settings, the use of self-signed certificates may be prohibited.

The installation of self-signed certificates is not supported by all web browsers. Depending on the web browser, it is possible to define exceptions.

For more detailed information, refer to the operating instructions of the web browser.

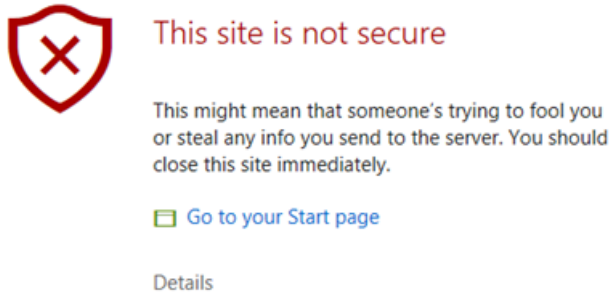
Installing the root certificate for Chrome and Microsoft Edge

Chrome and Microsoft Edge use the Windows system certificate store.

- On devices **with WinCC Unified installation** that have been configured with the Certificate Manager, these web browsers can immediately establish a secure connection to the WinCC Unified web pages because the root certificate has already been installed in the system certificate store.
- On devices **without WinCC Unified Installation** the root certificate must be installed manually.

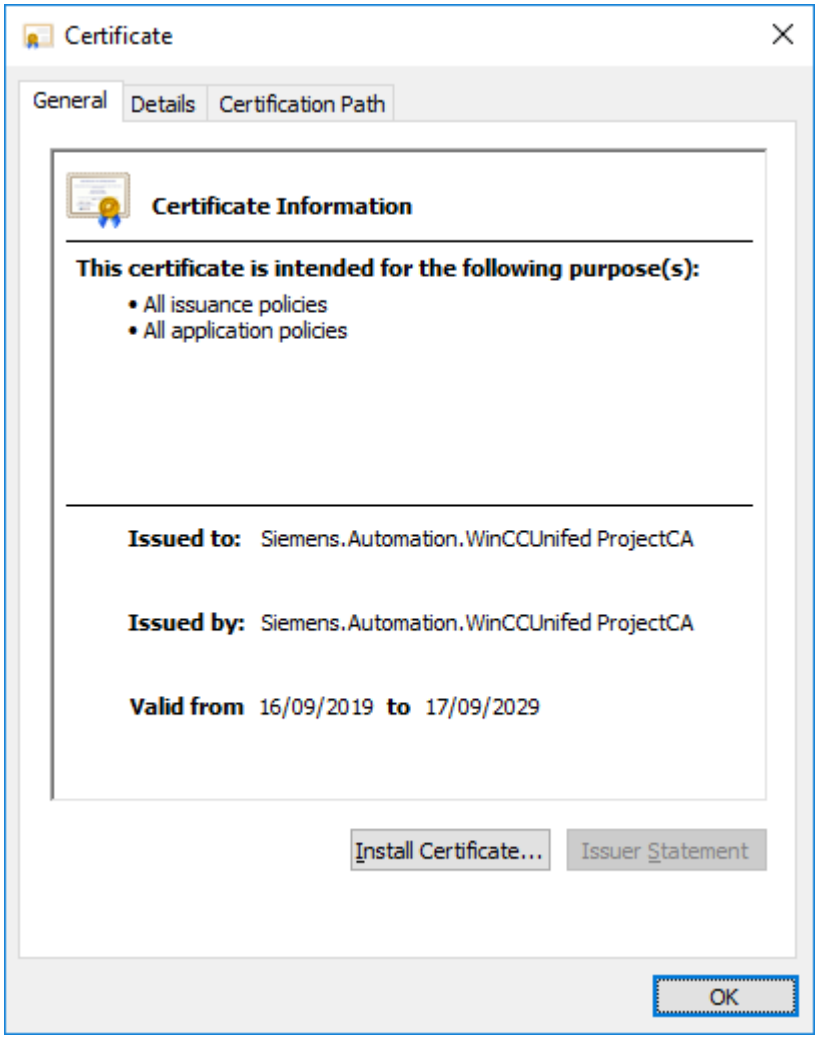
To install manually, follow these steps (for example, Microsoft Edge):

1. Open the WinCC Unified home page via the URL https://<host name>
   At first, an error message appears:



2. Open the field with the error details and confirm that you want to open the web page.
3. On the WinCC Unified home page, select the field "Certificate Authority" and confirm "Open file" in the download dialog.
   The root certificate is downloaded to the pre-selected download directory.

4. Open the downloaded file.
   The root certificate is opened with the Windows standard form.



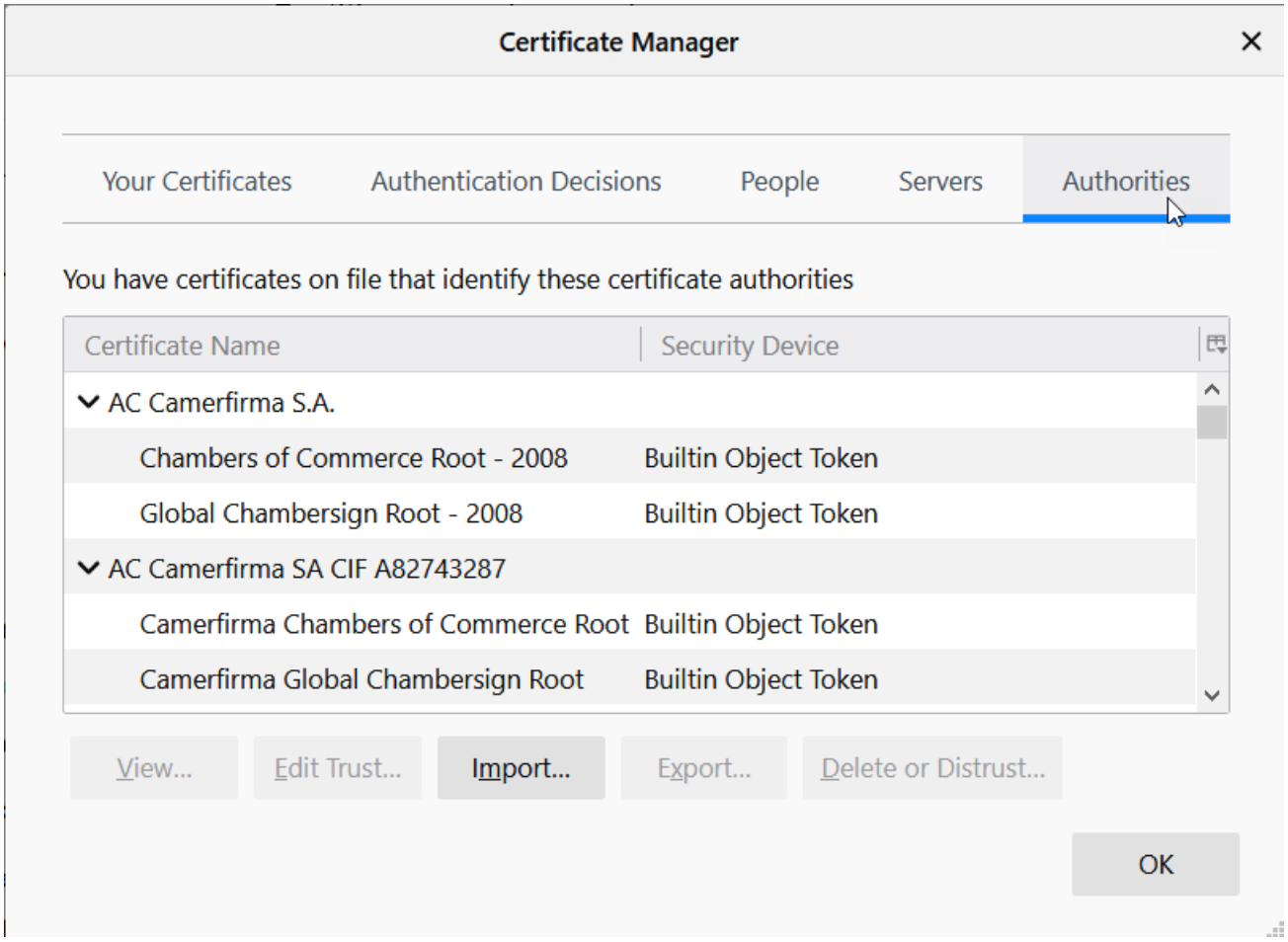
5. To import the root certificate into Windows, select "Install Certificate".
6. In the certificate import wizard, select "Local Machine" as the storage location, "Trusted Root Certification Authority" as the certificate store and start the import process.

Installing the root certificate for Firefox

Firefox uses its own certificate store and must therefore be configured manually on each device once:

1. Open the WinCC Unified home page via the URL https://<host name>
   At first, an error message appears:
2. Open the field "Advanced" and confirm the field "Accept the Risk and Continue".
   An exception is entered for this page in the Firefox certificate management.
3. On the WinCC Unified home page, select the field "Certificate Authority".

4. Save the root certificate. To do this, click "Save file" in the Firefox dialog that follows.
5. Store the certificate in the Firefox certificate store. Proceed as follows:
   - Open the "Settings" page of Firefox.
   - Select "Privacy & Security". There you will find the "Certificates" area further down. Open "Show certificates...".
   - In the "Certificate Management" window, select the "Certification authorities" tab:

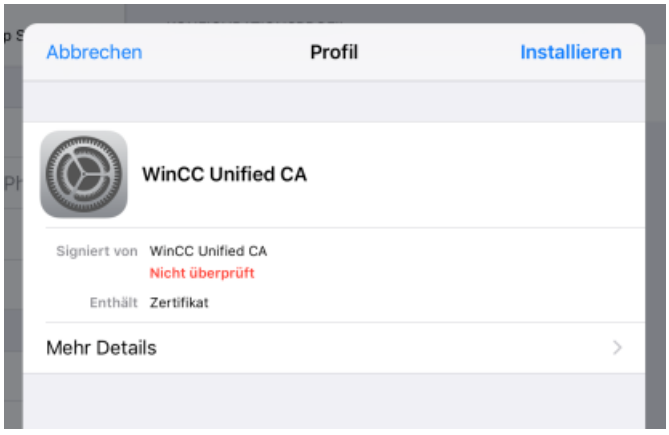


   - Click "Import" and select the root certificate you saved in step 3.
   - In the window that opens, select the option "This certificate can identify websites" and confirm your selection.
   - Click "Server" and remove the exception that was created by step 2.

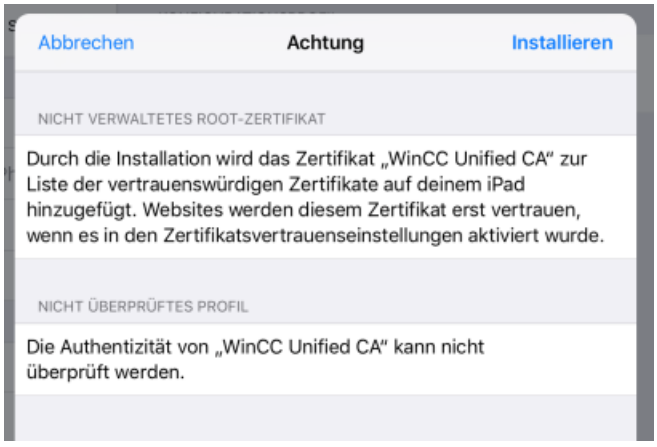
Installing the root certificate on iOS devices

iOS uses its own certificate store and must therefore be configured manually on each device once. An error message also appears when the WinCC Unified home page is opened.

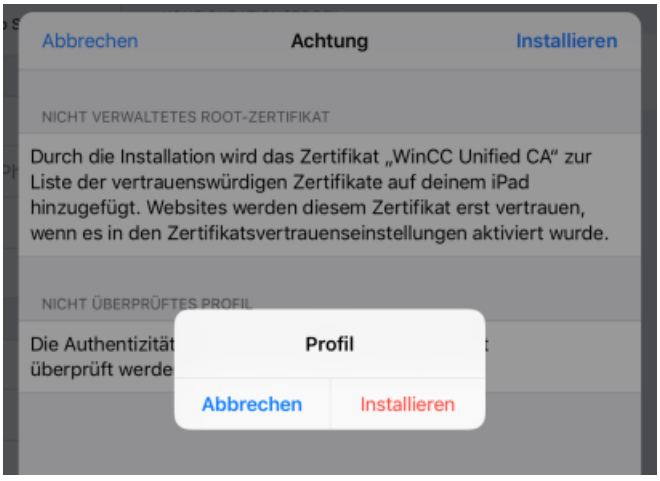
1. Open the field "Advanced" and confirm the field "Accept the Risk and Continue".
2. On the WinCC Unified home page, select the field "Certificate Authority".



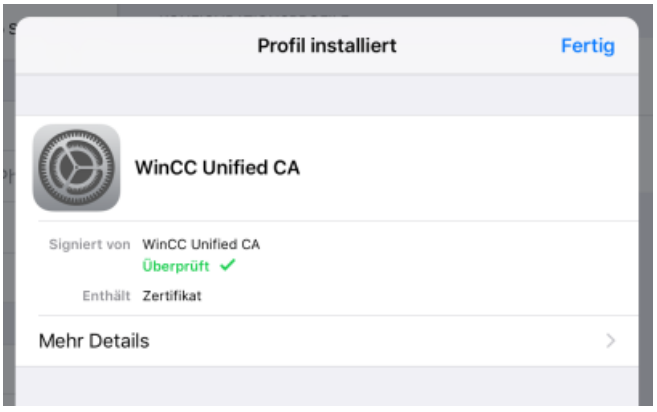
3. Select "Install".



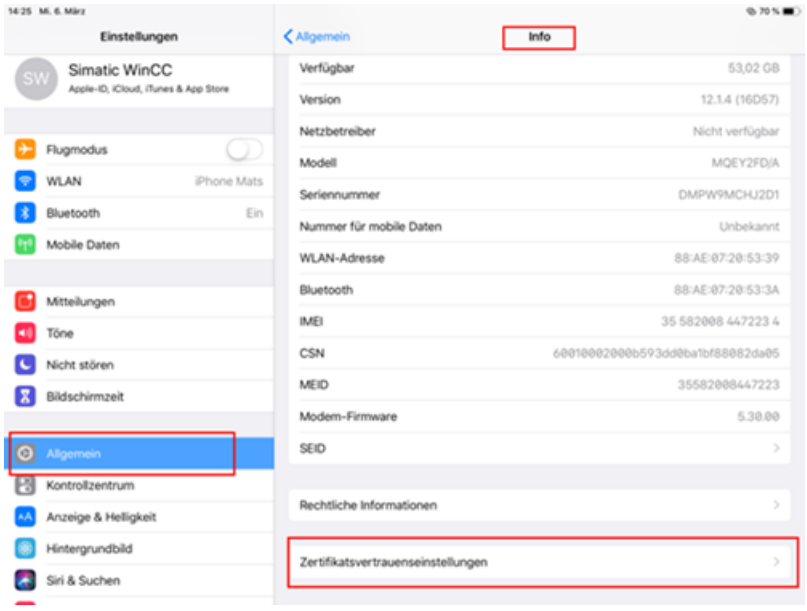
4. Select "Install" again.



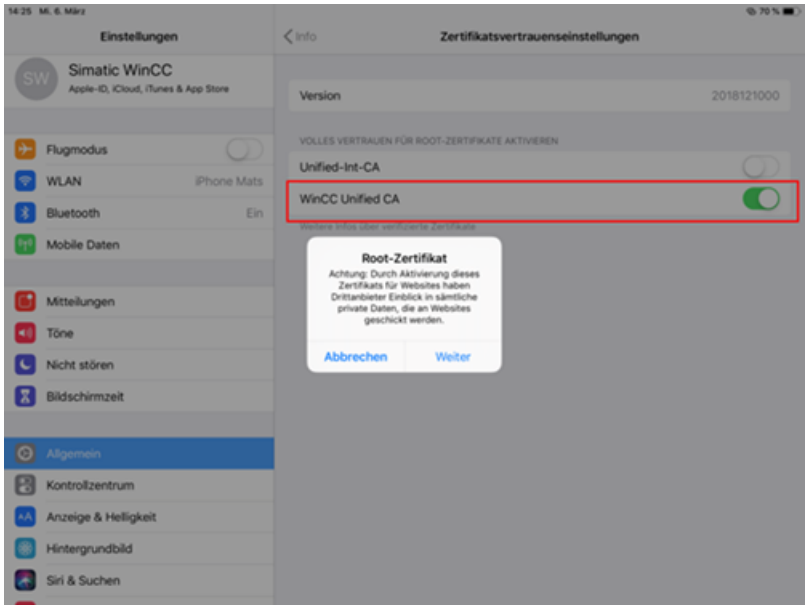
You see the entry "Trusted".



5. Select "General > Info > Certificate Trust Settings".



6. Activate "WinCC Unified CA" and select "Next".



See also

Using the web client (Page 7592)

18.3.6 Installing a certificate in the browser when accessing via web client (UCP)

Using the web server certificates

To enable web browsers to establish a secure connection to the Runtime of the Unified Comfort Panel, the certificate in the web browser must be known as trusted.

The procedure for installing the certificate differs depending on your web browser.

Use of self-signed certificates

When accessing the Unified Comfort Panel via web client, a self-signed certificate is used.

NOTICE
Security risk from self-signed certificate
A self-signed certificate is not issued by a trusted certification authority.
If you use a self-signed certificate from an untrustworthy source, the data transfer is not protected from attacks.
Before using self-signed certificates, check the source.
Depending on the firewall and network settings, the use of self-signed certificates may be prohibited.

The installation of self-signed certificates is not supported by all web browsers. Depending on the web browser, it is possible to define exceptions.

For more detailed information, refer to the operating instructions of the web browser.

See also

Using the web client (Page 7592)

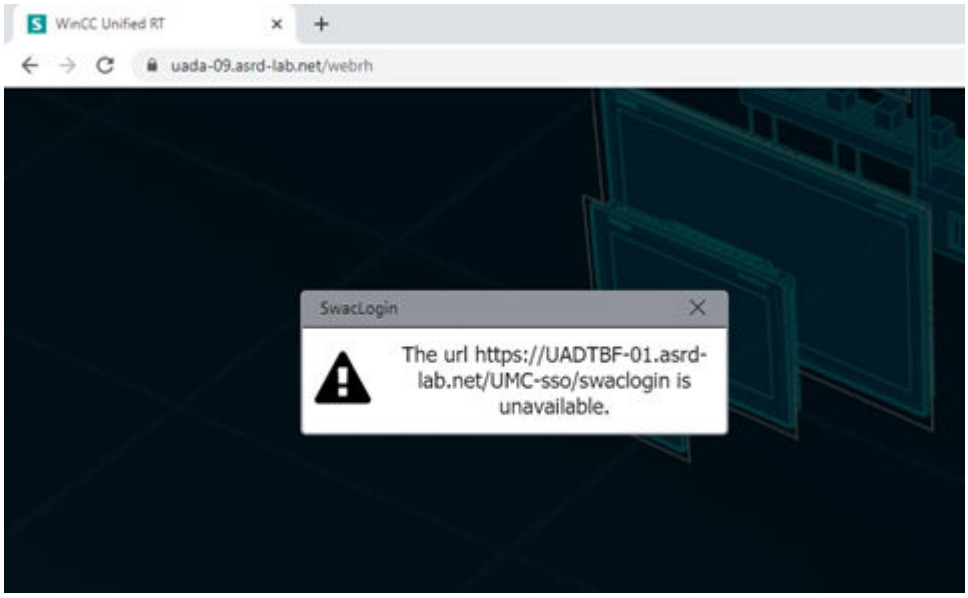
18.3.7 SwacLogin: Errors after complete download

After complete download of a project to a Unified PC, an error can occur when you open the WinCC Unified home page. The error can occur regardless of whether you open the home page locally on the PC or from a different device.

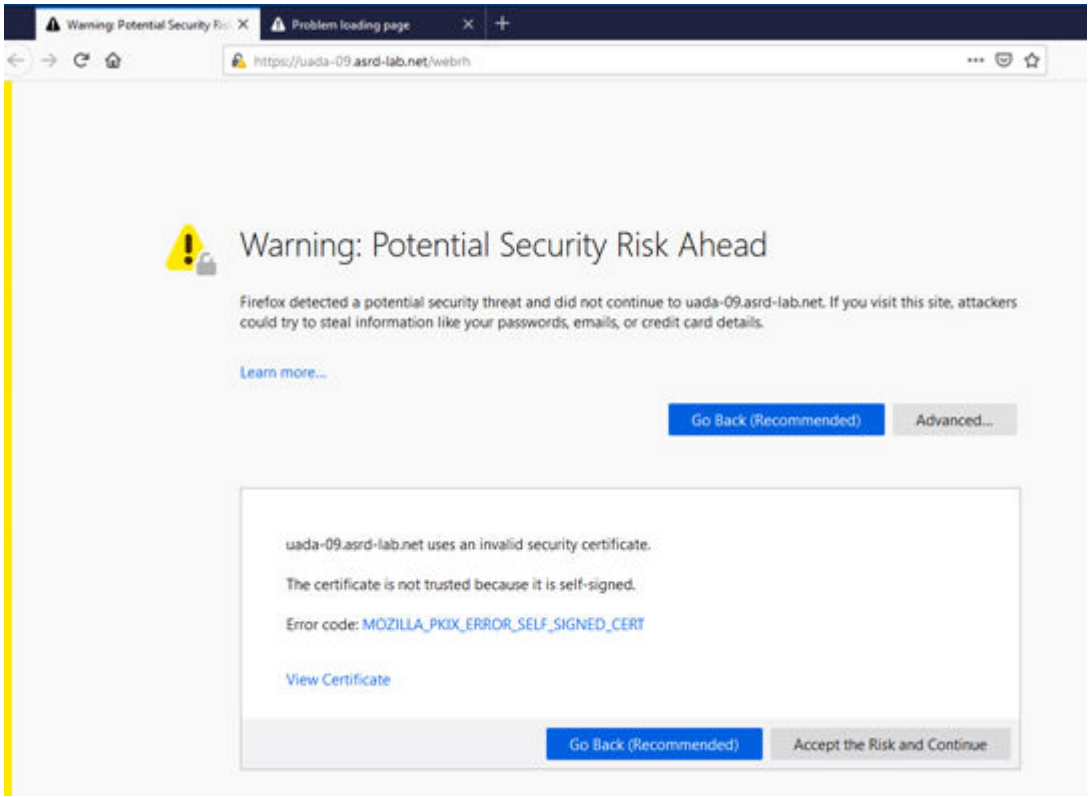
A possible cause of the error is the deletion of the browser cache.

Error description

In "Chrome" and "MS Edge", the error is displayed with the following alarm:



In "Firefox", the error is displayed with the following alarm:

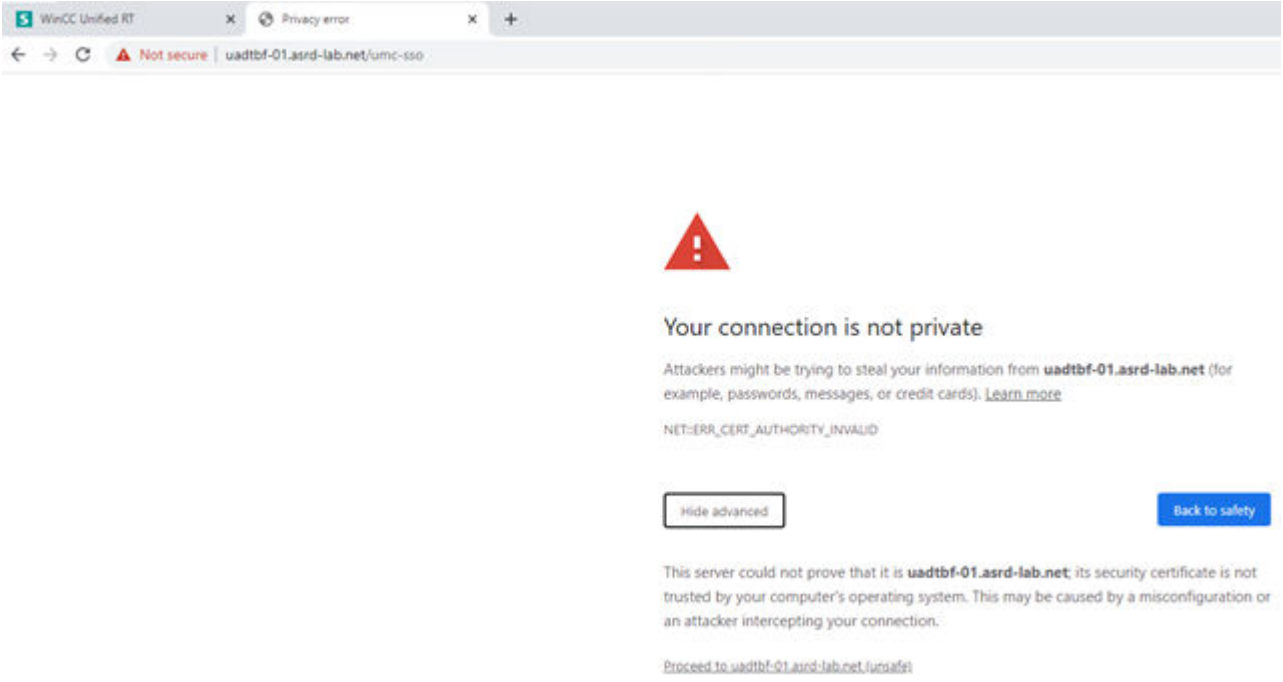


After accepting the warning of a potential security risk, the page remains empty in Firefox. Only the background screen is visible.

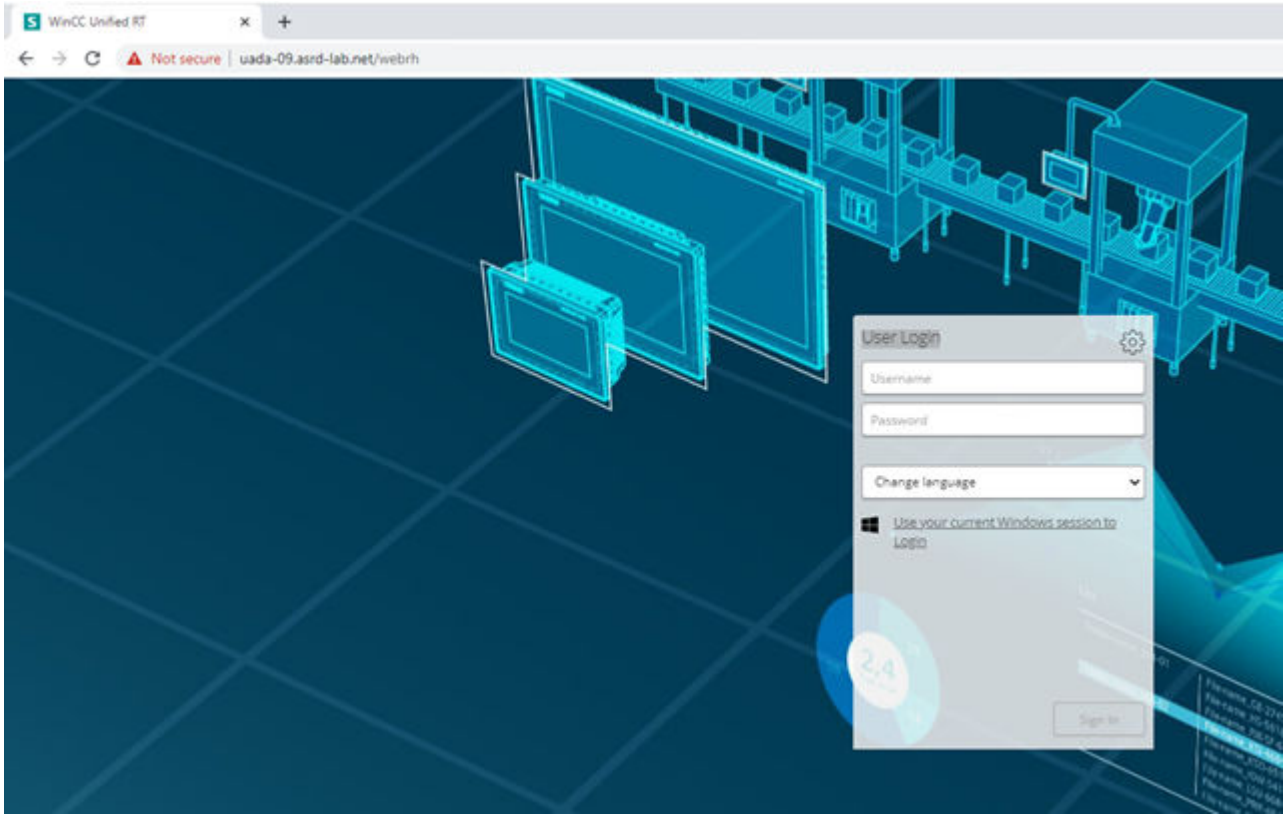
Remedy the error in "Chrome" and "MS Edge"

To remedy the error in "Chrome" and "MS Edge", follow these steps:

1. Open a new tab.
2. Enter the URL address of the identity provider of the UMC server in the address line of the browser. The URL is the same as the one in the error message without "/swaclogin", for example, "https://uadtbf-01.asrd-lab.net/umc-sso".
3. The page with a warning regarding the secure connection is displayed.



4. Accept the warning by clicking on "Proceed to uadtbf-01.asrd-lab.net (unsafe)".
5. The home page with the "User login" dialog is displayed.



Remedy the error in "Firefox"

To remedy the error in "Firefox", follow these steps:

1. Open a new tab.
2. Enter the URL address of the identity provider of the UMC server (ring server) in the address line of the browser, for example, "https://uadtbf-01.asrd-lab.net/umc-sso".
3. A blank page opens. Close the page.
4. Refresh the home page with the function key <F5>. The home page with the "User login" dialog is displayed.

See also

Using the web client (Page 7592)

18.3.8 Logging out user

If you want to end your Runtime session, you have the following options to log out completely:

- Use the "Logout" system function.
- Log out in the user management.
- Close all instances, i.e. open windows, of the browser in use.

Requirement

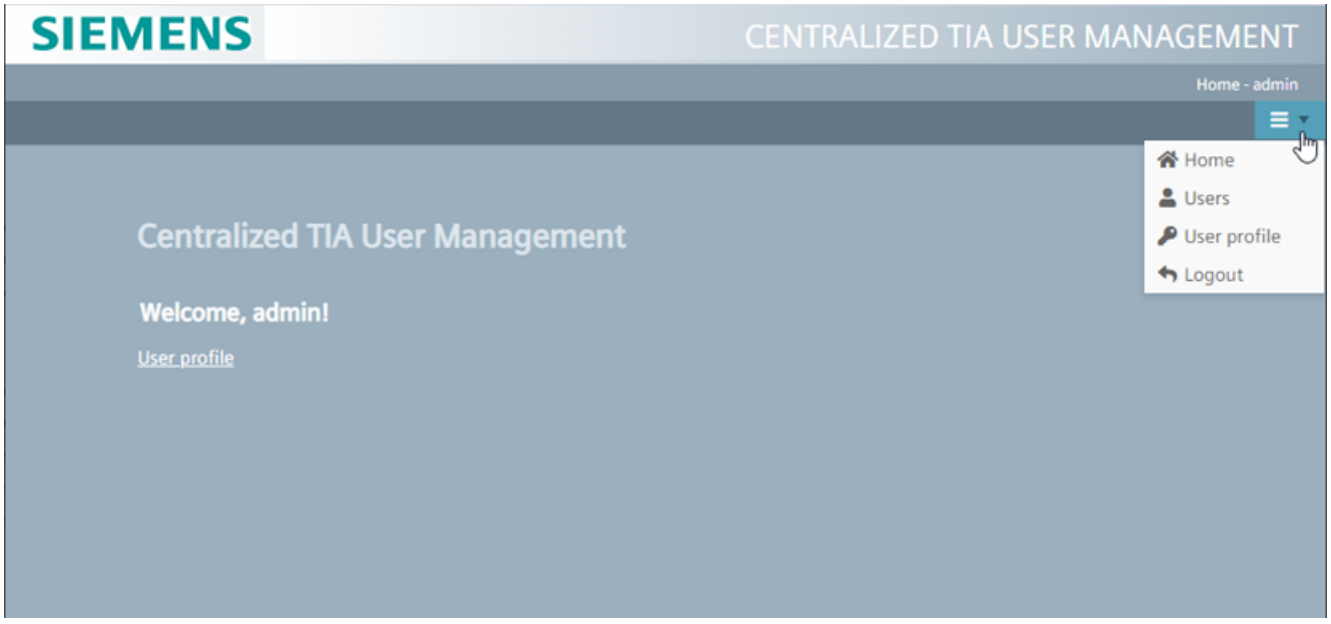
- You are logged in to Runtime.
- When you want to log out in the Runtime project, the system function "Logout" is configured, for example, to the event "Click left mouse button".

Logging out in the Runtime project with the system function "Logout"

- Select the button at which the system function "Logout" is configured.

Logging out in the user management

- Select "Logout" from the menu.
  Your session is ended.



New data downloaded from the TIA Portal is applied during the next login.

18.4 WinCC Smart Server

18.4.1 General

Introduction

With WinCC Smart Server, you can communicate between and with HMI systems via TCP/IP connections (e.g. LAN).

You can use the Smart Server in combination with SIMATIC Unified Comfort Panels.

Use of the SmartServer

You use Smart Server in the following application scenarios:

- Distributed HMI devices with Smart Client applications for controlling large machines or machines that are spread out over a large area.
- Remote control of an HMI device via the Internet or Intranet

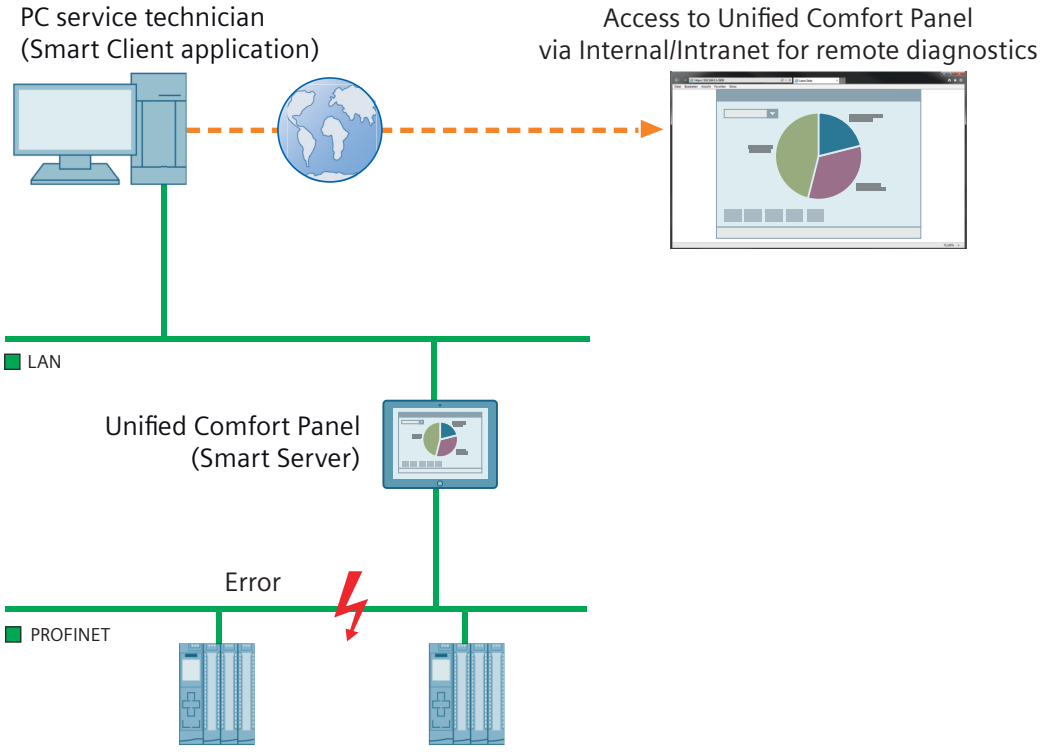
User benefits:

- Flexible solution for access to HMI devices from any location
- Reduction of load on the field bus:
  For example, Unified Runtime and Unified Comfort Panels provide a control system with access to process data. No load is placed by the factory level on the sensitive field level with respect to the necessary communication requirements. These requirements are handled by Unified Runtime as well as with the Unified Comfort Panels.
- Expensive on-site service visits to be avoided by using the remote control. Unplanned non-operation periods are reduced and the system productivity is increased.

18.4.2 Application scenarios

Remote diagnostics

A factory has a service contract with an external service company. The Unified Comfort Panel and the service technician's PC are linked together over a TCP/IP-ready network. The service technician accesses the Unified Comfort Panel via the Internet and executes remote diagnostics.



Application example

Among other things, flow rates are measured in the process control of a cooling unit. Contamination in a feed line reduces the flow of coolant. When the flow rate drops below the configured threshold value, the Unified Comfort Panel displays a warning.

The service technician then establishes a connection with the remote Unified Comfort Panel and takes the appropriate actions.

Advantage: The downtime is reduced to a minimum by remote maintenance.

Distributed HMI devices

Distributed devices, so-called Smart Clients, are used for controlling large machines or machines that are spread out over a large area.

The Smart Client establishes the connection to the Smart Server via the Smart Client application.

The operator can control and monitor the system with the Smart Client application and the Unified Comfort Panel. The operator sees the same screen with each Smart Client application and on the Unified Comfort Panel.

18.4.3 Security concept for the Smart Server

Remote monitoring and remote control of the Smart Server is protected by two functions:

- Encryption of communication with the Smart Server
- Passwords

Passwords for the Smart Server

Remote monitoring and remote control of the Smart Server is password protected. You create a password for each of two users.

- A password consists of exactly 8 characters.
- It contains at least one number (0-9)
- It contains at least one lowercase letter (a-z)
- It contains at least one uppercase letter (A-Z)
- It contains at least one of the following characters:
  !$%&()*+,-./:;<=>?@[\]_{|}~^
- Passwords cannot be entered using copy & paste.
- The passwords for User1 and User2 cannot be identical.
- Passwords must not be empty.

Note

Passwords for the Smart Server

No passwords have been preset.

You cannot use Smart Server if both passwords are not assigned.

Encrypt communication to the Smart Server

The option "Secure communication via self-signed certificates" enables an encrypted connection between Smart Client and Smart Server. When establishing the communication, the Smart Server sends a self-signed certificate to the Smart Client. The certificate must be accepted by the Smart Client.

Connections on Smart Client to Smart Server are then only possible on the basis of the exchanged certificate.

18.4.4 Settings in the TIA Portal

Introduction

In the "Runtime settings" editor, you configure the requirements for using the Smart Server.

As an alternative, configure the settings in the Control Panel of the Unified Comfort Panel.

Note that the settings on the Unified Comfort Panel are overwritten when a project is downloaded from the TIA Portal.

Requirement

- A project is open in the TIA Portal.
- A Unified Comfort Panel has been created.

Procedure

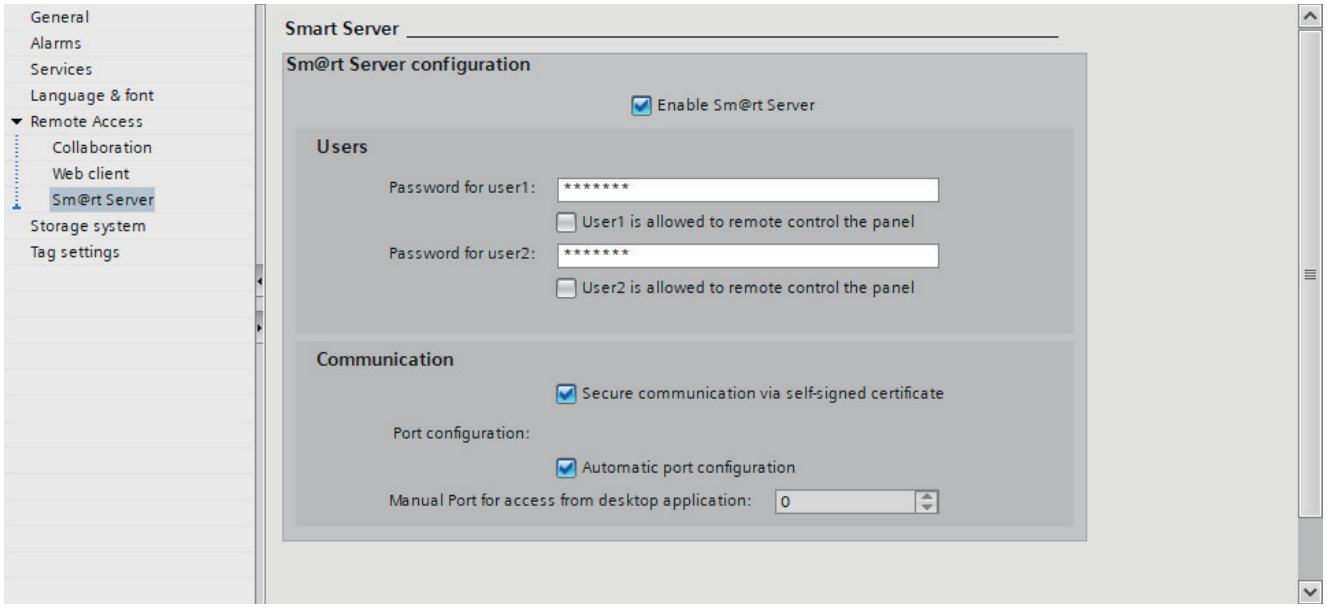
To configure the Smart Server, follow these steps:

1. Open the "Runtime settings" of the Unified Control Panels in the project tree.
2. In the "Runtime settings", expand the "Remote access" menu command.
3. Click "Smart Server".
4. Enable the "Enable Smart Server" setting in the "Smart Server Configuration" group.
5. Assign a password for "User1" and "User2" respectively.
   Note the Rules for passwords (Page 7608).
6. Activate "Secure communications via self-signed certificates".

   Note

   If "Secure communication using self-signed certificates" is not enabled, the communication is not encrypted.

7. If you do not want to use automatic port configuration, disable the option and assign a port manually.
8. Compile and download the project onto the Unified Comfort Panel.



18.4.5 Settings in the Control Panel of the Smart Server

The settings on the Control Panel of the Unified Comfort Panel regulate which remote operator is permitted to access the Smart Server application.

The options in the Control Panel are analogous to the runtime settings in the TIA Portal.

Note that the settings on the Unified Comfort Panel are overwritten when a project is downloaded from the TIA Portal.

Procedure

To configure the Smart Server in the Control Panel of the Unified Comfort Panel, follow these steps:

1. In the Control Panel of the Unified Control Panel, open the menu item "Network and Internet".
2. In the area "Remote access > Smart Server" enable the setting "Enable Smart Server".
3. Select one or more users for remote access.
4. Assign a password for each user.
   Note the Rules for passwords (Page 7608).
5. Activate "Secure communication via self-signed certificate".

   Note

   If "Secure communication via self-signed certificate" is not enabled, the communication is not encrypted.

6. If you do not want to use automatic port configuration, disable the option and assign a port manually.

18.4.6 Configuring the Smart Client application

18.4.6.1 Dialog "New SmartServer: Connection"

This dialog opens when you click the "SmartClient" button in the taskbar.

It is used to set the desired Smart Server and to select the connection method.

The following configuration options are available to you:

SmartServer

Enter the address of the server to which the connection is to be established. You can find the various options for entering the address under Remote control by means of the Smart Client application (Page 7613).

Connection profil

Select the type of connection to the Smart Server according to the network you are using.

Options

Opens the "Smart Client Options" dialog box with the technical settings for the Smart Client application.

18.4.6.2 "Options" dialog, "Connections" tab

The settings for the Smart Client application are specified in this dialog.

The following configuration options are available to you:

Format and encodings

Settings for compressing the screen data of the Smart Server.

- Use encoding
  Preassigned based on the selection under "Connection profile".
  Select the desired compression or "Raw" (no compression).
- Use 8-bit color
  Reduces the color depth at the client to 8 bits (256 colors). The data are then transferred faster. However, incorrect colors may result.
- Custom Compression level
  Allows individual customizing of the compression level in the "Level" input field:
  1 = least compression (faster); 9 = maximum compression (slower).
- Allow JPEG compression
  Allows the use of JPEG compression (involves losses).
  Enter the "Screen quality" in the input field underneath:
  1 = least compression (faster); 9 = maximum compression (slower).
- Allow CopyRect encoding
  Allows compression while using "similar rectangles".

Restrictions

- View only (inputs ingnored)
  Sets the view mode for this Smart Client irrespective of the settings on the Smart Server. This allows you to prevent unintended control operations.
- Disable clipboard transfer
  Turns off the clipboard.
  Applies only to the copying and pasting of texts.

Display

Settings for the screen display

- Scale by
  Zooms in or zooms out the desktop to be displayed.
- Full-screen Mode
  Displays the desktop to be shown in full-screen mode. If the display on the Smart Server is larger than the screen of the Smart Client, it is scrolled automatically by the mouse movement.

Mouse

Settings for the evaluation of mouse actions

- Emulate 3 Buttons (with 2-button click)
  Emulation of a three-button mouse by a two-button mouse.
- Swap mouse buttons 2 and 3
  Mouse buttons 2 and 3 are swapped.

Mouse cursor

Settings for the display of the cursor

Select the type of transfer of the mouse actions:

- Track remote cursor locally
  The information on the location of the cursor is transferred separately from the screen information. This speeds up the transfer of the cursor.
- Let remote server deal with mouse cursor
  Moves the Smart Server mouse pointer to the Smart Client mouse pointer. This allows more accurate cursor positioning.
- Don't show remote cursor
  The cursor at the Smart Server is not included in the transfer.

18.4.6.3 "Options" dialog, "Globals" tab

Technical settings for the Smart Client application are made in this dialog.

The following configuration options are available to you:

Interface options

- Show toolbars by default
  Shows the toolbar.
- Warn at switching to the full-screen mode
  Outputs a message before the full-screen mode is activated.

- Number of connections to remember
  The Smart Client creates a list of the recently used connections. This setting specifies the number of connections listed.
- Clear the list of saved connections
  The list is cleared.

Local cursor shape

Specifies the appearance of the local cursor. This makes it easier to differentiate between the local mouse pointer and the remote mouse pointer.

Logging

- Write log to a file
  Writes information to the logbook of the Smart Client application.
- Verbosity level

18.4.7 Remote control by means of the Smart Client application

Introduction

On the Smart Client, the Smart Client application provides the connection to the remote Unified Comfort Panel.

Requirement

- Both devices are connected with via a TCP/IP capable network.
- In the TIA Portal project of the Smart Server or in the Unified Comfort Panel, the setting "Enable Smart Server" is selected in the runtime settings under "Remote access > Smart Server".
  Note that the settings on the Unified Comfort Panel are overwritten when a project is downloaded from the TIA Portal.

Sequence

Remote monitoring or remote control is supported on the Smart Server.

Remote monitoring or remote control can be implemented on the Smart Client using the Smart Client application.

Remote control via the Smart Client application works as follows:

- Start the Smart Client application
- Establish connection
- Password input
- Operator control and monitoring on the Unified Comfort Panel

Install SmartClient application

The SmartClient application is executed via the "SmartClient.exe" file.

- If WinCC Runtime is installed on the SmartClient, the SmartClient application is also installed.
- If WinCC Runtime is not installed on the SmartClient, you have several options:
  - You copy the SmartClient application from the WinCC Runtime product DVD.
  - You copy the SmartClient application from the installation path from another PC.

Start the Smart Client application

- To connect to the remote Unified Comfort Panel, call the Smart Client application and enter the IP address of the Smart Server.
  - IP address or server name:port number
    - or -
  - IP address or server name:display number

  Example: "192.168.0.1::5800"

Other options to start the Smart Client application:

- Enter the name of the executable file and the IP address in the command line:
  `smartclient.exe 192.168.0.1`.
  The Logon dialog box appears.
  - or -
- Enter the name of the executable file, the IP address and the password in the command line:
  `smartclient.exe 192.168.0.1 /password <Password>`.

Password input

- Password input at the Smart Server
  Instead of the on-screen keyboard, the following message is displayed on the Smart Client when you enter the password directly at the Smart Server: "Remote access by Smart Options is in Progress. Please wait until the input of values has been ended." This measure prevents keyboard input for entering the password from being displayed on the Smart Client.
- Password input at the Smart Client
  The display of the screen keyboard on the Smart Server by actions on the Smart Client is suppressed. Use the local on-screen keyboard for entries at the Smart Client. The local on-screen keyboard on the Smart Client is automatically displayed. Close the on-screen keyboard manually. Select "Input > Hide Input Panel" to hide the local on-screen keyboard.

Note

The entries with full-screen keyboard are not protected on devices with a screen size of ≤ 6".

Entries in Control Panel Applets which do not use the full-screen keyboard are protected.

Note

Hidden password input is not supported by the on-screen keyboards of third-party products.

Note

It is not possible to enter special characters in combination with [AltGr].

Operation with the keyboard

For operator control via the keyboard, the following is available:

Keyboard shortcut	Function
<Alt+Ctrl+SHIFT+O>	Opens the "SmartClient options" dialog.
<Alt+Ctrl+SHIFT+F>	Switches over to full screen mode
<Alt+Ctrl+SHIFT+R>	Updates the display
<Alt+Ctrl+SHIFT+N>	Opens the "New SmartServer Connection" dialog
<Alt+Ctrl+SHIFT+S>	Save as
<Alt+Ctrl+SHIFT+T>	Displays and hides the toolbar

Result

The entire layout of the remote Unified Comfort Panel is shown in the application window. Depending on the configuration, you can specify monitoring only or operator control of all keys with the mouse.

18.4.8 Use and limitations of the Smart Server

Use restrictions

When using the Smart Server, observe the following notes:

- Smart Server and Smart Client
  - Use only simple projects.
  - Avoid photographs and color gradients in screens.
  - Avoid heavy background loads during operation, for example, those from user-defined functions or logs.
  - The maximum number of connected Smart Clients depends on the Unified Comfort Panel. For additional information, see the "Performance features" documentation.
  - To improve the performance of the Smart Server, you can disable the hardware acceleration of the graphics card.
  - For client access of a PC with touch screen to a SmartServer, the following applies to the touch operation of the SmartClient:
    Standard touch operation such as clicking or scrolling in lists is possible, but touch gestures and events such as "Press" and "Release" are not supported.
  - If you access a Unified Comfort Panel via the Smart Client, this leads to reduced performance on the HMI device.
- Access protection
  To protect access to an Unified Comfort Panel using different passwords, use the first password for protected access and the second password for unprotected access; for example, remote control with password and remote monitoring without password.
  Passwords must be assigned. It is not permitted to leave passwords blank.
- Port
  The connection to the web server is established via port 443 (HTTPS over SSL / TLS) or via port 80 (HTTP, unencrypted). For the web server to start without problems, make sure that port is not in use by any other application, such as the IIS World Wide Web Publishing Service.
- Timeout
  If the connection between the Smart Server and a Smart Client is interrupted, the server will register this disconnection only with a certain delay. The delay is based on the Windows standard configuration of TCP/IP Timeout.

Use-requirements in the company network

If the company network is protected by a Firewall, the system administrator must enable the corresponding port:

- The Smart Server is connected to the network via port 5900.

You change the port number in the Runtime settings under "Remote access > Smart Server" in the settings for "Communication" or in the same way in the Control Panel of the Unified Comfort Panel.

Options 19

19.1 WinCC Audit

19.1.1 Basics

19.1.1.1 GMP compliance

GMP-compliant projects with WinCC

Traceability and therefore the documentation of production data is becoming increasingly important in many sectors, for example

- Pharmaceutical industry
- Companies in the food and beverage industry or in the related mechanical engineering industry

Storage of production data in electronic form offers many advantages compared to paper documents, such as simple acquisition and logging of data.

Events that are relevant for monitoring and ensuring the correctness of the process occur during runtime. Although the data provides information about a specific action, it is complicated for the user to easily identify the relevant data in the data set, such as: Cause of the change in value, actions that led to the change, or additional details about the change itself (e.g. source, location, cause).

However, it is also important to ensure that data cannot be falsified and that it can be read at any time.

Therefore, sector-specific and cross-industry standards have been developed for the electronic documentation of production data.

The most important set of regulations is the FDA Guideline 21 CFR Part 11 for electronic data records and electronic signatures issued by the FDA, the US Food and Drug Administration. In addition, different EU regulations apply, for example, EU 178/2002, depending on the industry.

Requirements for production systems in these industries have been developed on the basis of 21 CFR Part 11 and the corresponding layout to comply with GMP (Good Manufacturing Practice). They are also required for other industries.

The following primary requirements are derived from these directives and rules:

- Creation of an Audit Trail or operating trace in runtime
  Based on this document, it is possible to trace the user who carried out an operator action on the machine at what time.
- Important process steps must also be assignable to a clear responsibility, for example, via an acknowledgment or a comment.

19.1.1.2 GMP-compliant configuration

Introduction

"Configuration conforms to GMP" means creating projects in accordance with "Good Manufacturing Practice". The requirements are set out in FDA rules "21 CFR Part 11". The FDA is the U.S. Food and Drug Administration. Eudralex Volume 4, Appendix 1, EMA regulation 178/2002 also applies. EMA is the European Medicines Agency.

GMP-compliant configuration means HMI devices have electronic production data documentation functionalities.

Note that the WinCC Audit option is currently only available for Unified SCADA.

GMP-relevant and Audit Trail

WinCC offers the "Audit" option for implementing GMP compliance. Using the Audit option, the "Configuration conforms to GMP" function can be enabled.

Enable the "Configuration conforms to GMP" function directly in the runtime settings of the HMI device. GMP relevant functionalities are then added to WinCC. These functionalities are:

- Option to label tags as "GMP relevant".
- Suggestions for the comment with typical reasons for changes of GMP-relevant tags can be pre-defined by selecting text lists.
- Automatic identification of significant changes in GMP-relevant tags
- Generation and storage of electronic records of the relevant changes - Audit Trail
- System function for recording relevant user actions - electronic recording
- Recording of manipulated log data with checksum
- Audit trail record for printing logged changes

Execution of or changes to labeled objects are saved in a special log, the "AuditTrail".

Licensing

To use the GMP relevant functionalities configured in WinCC in runtime, you need the following license:

- WinCC Audit for RT Unified

19.1.1.3 Audit option

Advanced functions

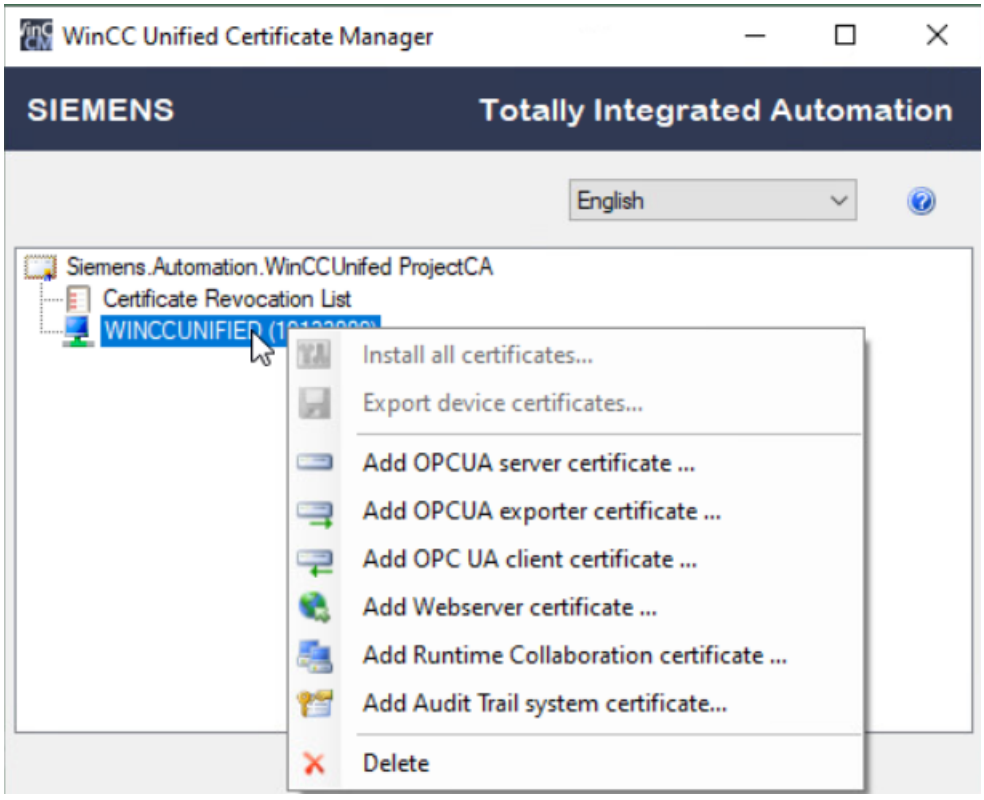
The Audit option adds functions to WinCC to ensure that your project is GMP compliant.

The following functionalities are added:

- Audit Trail
  For each HMI device, you can create one Audit Trail .
  Operator actions and system processes that are relevant for the FDA compliance of the process are recorded in an Audit Trail during runtime.
  - Actions by the system, such as starting up runtime or rejecting logon attempts.

Audit Trail certificates in Runtime

Audit Trail certificates are required for the HMI device. The certificates are used to check integrity and prevent tampering with the archive. To install the certificate on the HMI device, use the WinCC Unified Certificate Manager. Note that the maximum lifetime of a certificate is 150 months.



Note

Note that an error message appears in the alarm control without the certificate or with an expired certificate.

Extension of the WinCC engineering system

For HMI devices that support "Configuration conforms to GMP", the WinCC engineering system is extended to include the following configuration options when GMP is enabled:

- The Audit Trail entry is added to the "Logs" editor
- A "Good Manufacturing Practice Settings" entry is added to the "HMI tags" editor in the Inspector window of a "Properties > Properties" tag
- "InsertElectronicRecord" system function

See also

Configuring the "InsertElectronicRecord" system function (Page 7637)

19.1.1.4 Scope of logging

It is important to ensure that Audit-related processes are always logged in runtime in the Audit Trail in a project with the option "Audit". Logging of the Audit Trail depends on the device used.

You have two options for logging your data in WinCC Unified:

- File-based logging
  File-based logging allows you to log up to 5000 logging tags in an SQL Lite database.
  You do not need a license to log logging tags in WinCC Unified PC.
  You need a WinCC Unified Runtime (RT) license to log logging tags in WinCC Unified PC.
- Database-based logging
  Database-based logging allows you to log all logging tags up to the high limit in an MS SQL database.
  Database-based logging is only available for WinCC Unified PC, and you need a WinCC Unified Runtime (RT) license to log logging tags.
  Besides the functionality, database-based logging also includes an MS SQL server. Therefore, you need the "WinCC Unified Database Storage" license.

Scope of logging

The following Audit-relevant processes are logged depending on the configuration of the tags of the project:

- Value changes of GMP-relevant tags by the user
- "InsertElectronicRecord" system function
  You use the "InsertElectronicRecord" system function to record user actions that are not automatically recorded by the Audit Trail.
  You can configure this system function to screen calls, for example, or you configure function lists containing system functions that do not require acknowledgment.

19.1.1.5 Performance features of the GMP-compliant configuration

Supported HMI devices

Supported HMI devices

The qualification "GMP relevant configuration" can be configured for the following HMI devices:

- Unified PC
- Unified Comfort Panel

Restrictions

Restrictions

The following functions and configurations cannot be used simultaneously with the qualification "GMP relevant configuration":

- PN direct keys
- DP DirectKey
- Events of screen objects
  You can set important user actions as GMP-relevant in runtime, such as changing tag values. In this case, you may not assign any other events to this graphic object.
  When the event of a screen object is assigned actions which open a user dialog, you may not be able to execute these actions at other events.
- Controlling GMP-relevant tags using a slider
  The slider is not suitable for controlling GMP-relevant tags. Any operation of the slider will continuously change the tag value. If this is a GMP-relevant tag, a flood of entries is generated in the "AuditTrail".

19.1.2 Using the Audit trail

19.1.2.1 Enabling GMP compliant configuration

Introduction

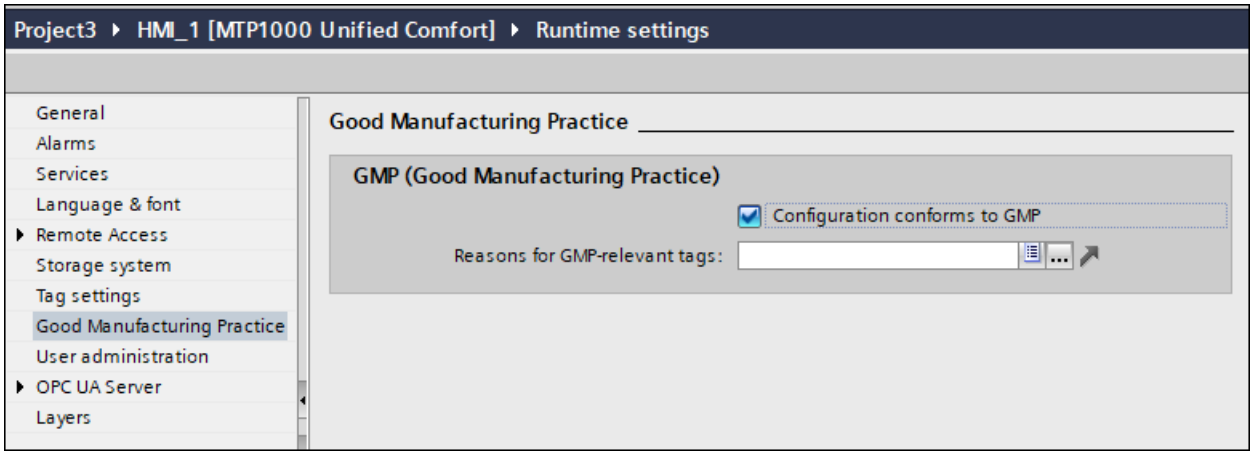
The function is made available to the Audit Trail by labeling it "Configuration conforms to GMP".

Requirements

- A project is created.
- A GMP compatible HMI device has been created.

Procedure

1. Click on the HMI device in the project tree.
2. Double-click on "Runtime settings" in the project tree. The editor opens.
3. Click on "Good Manufacturing Practice".
4. Select "Configuration conforms to GMP".



Result

The Audit option is now enabled for the HMI device.

The following functions can now be configured:

- Audit Trail log
  - For each HMI device, you can create one Audit Trail
- "InsertElectronicRecord" system function
- GMP-relevant tags
- Reasons for GMP-relevant tags (available text lists with pre-defined comments for acknowledgment in Runtime)
- GMP relevant recipes

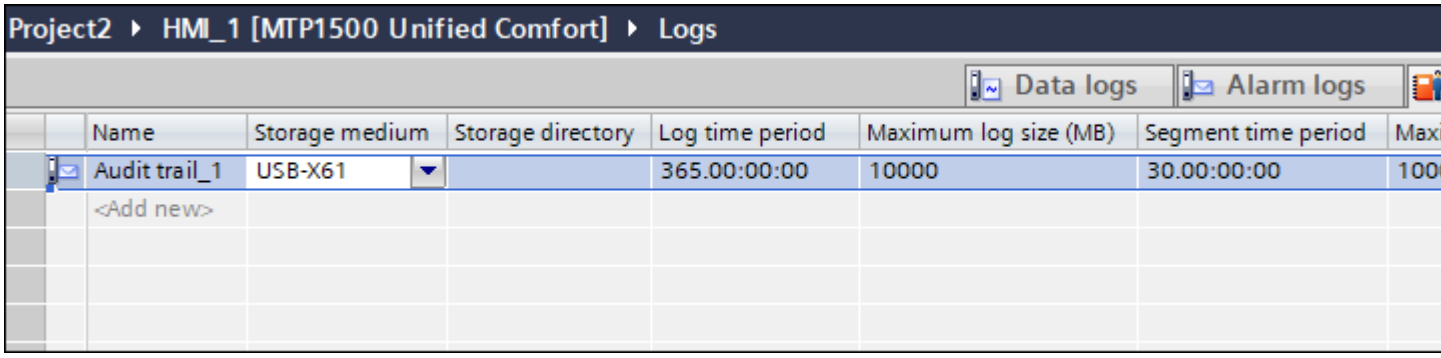
19.1.2.2 Creating an audit trail

Requirements

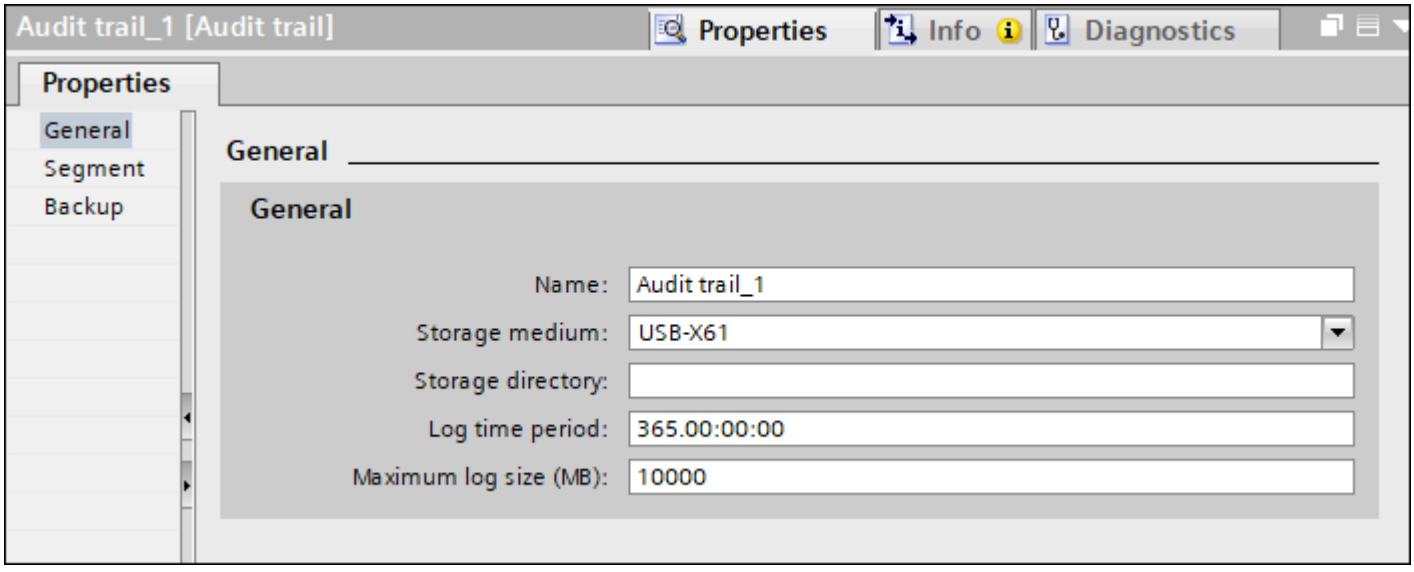
"Configuration conforms to GMP" has been selected on the HMI device. You can only create one Audit Trail per HMI device.

Procedure

1. Click on the HMI device in the project tree.
2. Double-click "Logs".
   The "Logs" editor will open.
3. Change to the "Audit Trail" tab.
   An Audit Trail has been created.



4. Set the parameters of the "AuditTrail" in the Inspector window under "Properties > Properties > General".



Storage medium and storage directory

The following settings are available for the storage medium and the storage directory:

- Default: You select the storage location that is defined in the WinCC Unified configuration on the runtime machine as storage directory.
- Local: Under storage directory, enter the path to the storage location. Make sure that the user has write permission.
- Project folder: As storage directory, you select the runtime folder to which the project data is copied after the download.

Log time period

The standard time period is 365 days. You can specify the log time period in the following format: Days.Hours:Minutes:Seconds.

Maximum log size

The standard log size is 10 GB. The value is entered in MB. When the maximum log size has been reached, the oldest entries are overwritten.

Log segments

The size of the log segments is specified either as a time period or a segment size. You specify the following under "Properties > Properties > Segment":

- Segment time period: The standard time period for logging a single segment is 30 days. You can specify the time period in the following format: Days.Hours:Minutes:Seconds.
- Maximum segment size: The standard segment size is 1 GB. The value is entered in MB.
- Segment start time: Specifies the start time of the log segment.

Backup

When the databases exist in the MSSQL format, you can create backups of the database under "Properties > Properties > Backup".

1. Under "Backup mode", select the "Path" setting.
2. Enter a storage location under "Backup path".

19.1.2.3 Audit Trail reports

Basics

Based on an Audit Trail report, it is possible to trace the operator who carried out operator actions at what time. The configuration engineer uses the WinCC Unfied Excel add-in to configure a report template with Audit Trail parameters that are relevant for quality assurance in the manufacturing process. You can print out the report either as an Excel file, a PDF file or send it via email.

General information on configuring report templates in the Excel add-in is available in Creating report templates for production protocols (Page 7655).

General functions on the configuration of report jobs in runtime are available in Working with production protocols in runtime (Page 7719).

Audit Trail parameters

The following table contains an overview of the potential values of the individual report columns:

Parameter	Description
Time stamp	Contains the time stamp at which the event occurred.
Object name	Contains the name of the object that triggered the change.
User	Contains the name of the logged-on user. When no user is logged on, the field remains empty.
Operator Station	Contains either the name of the PC, PLC or the IP address of the web client.
Old value	Contains the tag value before the change.
New value	Contains the tag value after the change.
Cause	Contains the user comment. Contains the confirmation of the desired state.
Event ID	Contains the internal identifier of the event.
Tracking ID	Contains the internal identifier to link the user action to the system response.
Provider type of the audit	Contains an identifier for the type of intervention: 2 - Task scheduler 6 - Operator action 11 - System response
Audit provider	Contains the name of the provider, for example, Scheduler, User Interface, Event Manager.
Type of operation	Contains an identifier for the change: 1 - New value 2 - Updated value 3 - Deleted value
Object reference	Contains an identifier of the triggering object. This parameter is reserved for internal purposes.
Integrity	Contains an identifier for proof that data was manipulated later.

See also

Add Audit (Page 7424)

Adding or editing configurations for audit (Page 7434)

Add Audit

Introduction

To output the Runtime device Audit Trail in a report, add an Audit data source item to a report template.

You can find more information about the Audit option in WinCC Unified in the TIA Portal help.

Requirement

- The Audit option was activated in the engineering for the Runtime device.

- The "Audit" option is activated in the connection settings of the Excel add-in.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a time series segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Audit" option.
5. Select the Audit Trail.
6. (Optional) To undo your selection, select the Audit Trail under "Selected data source items" and click "Delete".
7. Confirm with "OK".

Result

The Audit data source item is displayed below the segment.

If an Audit Trail is configured for the data source, the Audit data is inserted into the report when the Runtime data is read into Excel and when it is generated in Runtime:

- In the legend table: Identifier of the overall status of the Audit Trail for the queried time range in the "Audit Status" field

Value	Description
Green	No manipulations of the Audit Trail were found in the queried time range.
Red	Manipulations of the Audit Trail were found in the queried time range. Single or multiple entries have been deleted, added or changed.

Requirement: The "Audit status" option is activated on the segment under "Header properties".

Note

Overall status for check mode "None"

If the check mode "None" is set in the configuration of the audit data sources item, the "Audit status" field is always green.

- In the data table of the segment: Identifier of manipulations

Type of manipulation	Identifier in the data table
Value of entries changed	Directly at the entries
Entries added	
Entries deleted	The manipulated time range receives a start and end entry.

First, the data table shows the contents configured in the standard configuration for Audit. To output other contents, select or create a configuration.

Adding or editing configurations for audit

Introduction

Check mode

The check mode of the configuration of an audit data source item determines

- whether an integrity check is performed when the runtime data is read, and what is checked. You can output the overall result of the check in the table header row in the "Audit status" field.
- Which audit data records are provided in the data table.

Possible check modes:

"None"	Provides the data for all audit data records that fall within the requested time range. No integrity check is performed. Default setting
"Check only"	Checks all audit data records that fall within the requested time range without providing their data. It is tested whether data records have been manipulated, deleted or added.
"Check entries"	Checks the audit data records that fall within the requested time range and that have not been deleted from the audit trail or subsequently added, and provides their data. It is checked whether data records have been manipulated.
"Check all"	Checks all audit data records that fall within the requested time range and provides their data. It is tested whether data records were manipulated, deleted from the audit trail or subsequently added.

Filter type

An Audit data record consists of two entries:

- An entry for the user expectation
- An entry for the system response.

User expectation and system response may differ. In addition, there are cases in which only one of the two entries is created.

The filter type controls which data records and which entries are included in the report.

Possible filter types:

Filter type	**User expectation equals system response**	**User expectation does not equal system response**	**Data record entry for user expectation or system response is missing**
"Show all data in detail"	Both data record entries are inserted.		The existing data record entry is inserted.
"Show data and conformity errors"	The data record entry with the user expectation is inserted.	Both data record entries are inserted.	
"Show only data with conformity errors"	No data record entry inserted.		

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration".
3. Click "New Configuration > Audit configuration".
4. Enter the name of the configuration under "Name".
5. Select a check mode:

6. Determine which audit attributes are displayed. To do this, activate the options for the desired columns under "Columns".
7. Specify a filter type.
   Preset value: "Show data and conformity errors"
8. (Optional) To further filter the inserted content, define a filter query.
   The filter query can consist of up to two conditions. Follow these steps:
   - Under "Filter query", click "+" or "Add new condition row".
   - Select an Audit attribute, an operator and enter the value of the attribute.
   - Optional: Use "+" or "Add new condition row" to create further conditions and select whether the conditions are to be linked with a logical AND or OR.
9. Confirm your entries with "OK".

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration".
3. Click a configuration for Audit.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Examples of the configuration of the filter type

The following table contains examples of data records that were generated in Runtime through changes to tags monitored by Audit:

Data record ID	Tag name	Modified by	Old value	New value	Description
1A	Motor1_Speed	User1	0	10	An operator changes the speed of a motor in an I/O field of an HMI screen.
1B	Motor1_Speed	System	0	10	User expectation and system response are identical.
2A	ValvePercentile	User1	0	100	An operator opens a valve using a slider on an HMI screen. The valve has a physical blockage and cannot be opened. Therefore, no data record entry for the system response is generated.

Data record ID	Tag name	Modified by	Old value	New value	Description
3A	ValvePercen-tile	User1	0	99	A physical blockage has been removed and the operator repeats the entry. The valve reacts, but cannot be fully opened. User expectation and system response differ.
3B	ValvePercen-tile	System	0	49	
4B	Mo-tor2_Speed	System	0	20	An operator changed the speed of another motor. The resulting data record was manipulated, and the user expectation entry was deleted. There is only one entry for the system response.

The following table shows which data record entries are inserted into the data table depending on the filter type selected when generating the report:

Data record ID	Tag name	Modified by	Old value	New value
Filter type "Show all data in detail"				
1A	Motor1_Speed	User1	0	10
1B	Motor1_Speed	System	0	10
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20
Filter type "Show data and conformity errors"				
1A	Motor1_Speed	User1	0	10
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20
Filter type "Show only data with conformity errors"				
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20

19.1.2.4 Audit trail logging concept

Format

Audit Trail format

On an HMI device with "Configuration conforms to GMP", all events that are relevant to the Audit are recorded in runtime in the Audit Trail. You have several format options.

Selection is dependent on the display program and the runtime language used:

- RDB file
  Data is saved with quick access in a proprietary database.
  If you require maximum read performance in Runtime, use the "RDB file" storage location.
- csv file
  To view and evaluate a csv file, use Microsoft Excel or similar on your PC.

  Note

  Double quotation marks or several characters are not permitted as list separators for the format "File - csv (ASCII)". You can find the settings for list separators under "Start > Settings > Control Panel > Regional and Language Options".

- TXT file
  This file format supports all characters that can be used in WinCC. For editing, you will need software that can save files in Unicode, such as Notepad.

  Note

  Use "File - TXT (Unicode)" to log Asian languages.

Audit Trail with checksum

Audit Trail with checksum is automatically generated in the database that is based on the user certificate. The certificates are created with the WinCC Certificate Manager Tool. You can find additional information in the Runtime online help in the Certificate Manager section.

Storage location and medium

Storage location and medium

Depending on the hardware configuration of the HMI device, the data may be logged locally (on the hard disk of a PC or on the storage card of a panel) or, if present, on a network drive.

Note

Logging on network drives

We do not recommend that you log Audit Trails directly on a network drive. Power supply can be interrupted at any time. This means there is no guarantee for a reliable operation of logs and Audit Trails.

We recommend you save the logs on your local hard drive, or on a storage medium of the HMI device. Use the system function "ArchiveLogFile" to save the logs long-term on a network drive.

"Configuration conforms to GMP" can only be fully operated in runtime as long as Audit-relevant user actions can be saved in the Audit Trail. It is important to ensure that sufficient memory space is available for the Audit Trail and that the connection to the storage location of the Audit Trail is not interrupted.

Error-handling with insufficient free storage space

If there is insufficient storage space, your project can be configured so that the administrator has an option of continuing the process without logging in the Audit Trail (forcing).

Error-handling if there is no storage medium or the connection to the server is interrupted

If the storage space for the Audit Trail is insufficient, for example, if there is no storage medium, all Audit-relevant user actions are blocked.

The block is canceled as soon as the storage location for the Audit Trail can be reached again. The block can be skipped by "forcing".

Error-handling with long-term logging

If the Audit Trail must be moved to a server for long-term logging and the connection to the server is interrupted at this time, the following error-handling is required:

The system closes the Audit Trail and renames it. The system attempts to send the renamed Audit Trail to the server again in the background.

If disruption in the connection to the server persists, you receive a system alarm telling you that the connection is down. Then the system attempts to send the renamed Audit Trail every 300 seconds.

The attempt to transmit the data is repeated until successfully completed. The data is also transmitted after a restart of the HMI device.

Protection mechanisms

Protective mechanisms to prevent changes to Audit Trail data

The Audit Trail data are protected against deliberate or accidental changes:

- The directory in which the Audit Trail is saved can only be accessed with special rights.
- The Audit Trail files are write-protected.
- In addition, electronic records that have been removed or added in a certain time segment are identified.
- Each data record contains a checksum that can be used to detect a change of its contents. The checksum is generated based on a certificate. This checksum also ensures that the number of lines has not changed in the Audit Trail file.

Use the "HmiCheckLogIntegrity" tool, included in the Audit option, to check whether an Audit Trail has been changed:

Upgrading WinCC

Upgrading WinCC

Before you update WinCC with a Service Pack or a new version, you will have to exit and save the Audit Trail or the logs with checksum. After WinCC is updated, the audit trail or logs with checksum will be continued with new files.

Make sure that the logs are started at a defined state with the new version.

Audit trail behavior in runtime

Effects in runtime

The configuration in Audit Trail has the following effects in runtime, depending on the configuration:

- Audit-relevant user actions (such as tag changes) are recorded in an Audit Trail.
- For GMP-relevant tags, the system automatically generates an electronic record as an action requested by the user (requested action) and an electronic record of the system reaction (performed action).
- "Enable logging at runtime start" check box enabled:
  The Audit Trail is started with runtime.
- "Forcing" group, "Allowed if storage space has been exhausted" check box enabled:
  A user with administrator rights can use "forcing" to run operations on the plant even though the Audit Trail can no longer be logged because of storage space limitations. Interrupting the Audit Trail prevents the process from being stopped.
  If the check box "Signing may be bypassed" is enabled, the administrator is not required to input electronic signatures, acknowledgments or comments for operator actions that would normally require signing, acknowledgment or comment.
- Depending on the configuration, the texts can be selected from a pre-defined text list for acknowledgment in Runtime.
- If the storage space available for the Audit Trail is less than the configured "Free storage space limit in MB", the function list configured for the "Low free space" event will be processed.
- If there is insufficient storage space for the Audit Trail because of hardware limits, the function list configured for the "Free space critically low" event will be processed.

19.1.3 Configuring audit functions

19.1.3.1 Logging tag value changes

GMP-relevant tags

GMP-relevant tags

To log changes to the tag value "Good Manufacturing Practice" (GMP), mark a tag as GMP relevant. Special consideration should be given to the following:

- Array tags: When you mark an array variable as GMP relevant, all elements are automatically labeled as GMP relevant. You cannot change the setting of the individual elements.
- User data types: When you mark a user data type as GMP relevant, all elements are automatically labeled as GMP relevant. You cannot change the setting of the individual elements.
- System tags: System tags cannot be marked as GMP relevant.

Changes to tag values

When a tag is marked as GMP relevant, changes to the tag value are logged in an Audit Trail.

- Value changes by a user are logged in the Audit Trail.
- Value changes caused by a system function that is configured to an event are logged when the system function is triggered by a direct user action.
- Value changes that are made by the PLC or a system function are not logged in the Audit Trail.

Effects in runtime

The configuration has the following effects in runtime depending on the properties of the GMP-relevant tags:

- If the user changes the value of a GMP-relevant tag in runtime, the value change is entered in the Audit Trail.
- Acknowledgment
  If "Acknowledgment" is specified as the "Type of confirmation", the user must acknowledge a value change of this tag. Otherwise, the value change is rejected.
  The acknowledgment is logged in the Audit Trail.
- Comment
  If the "Comment required" setting is selected in addition to the acknowledgment, the user must acknowledge and comment a value change of this tag. Otherwise, the value change is rejected. The user can enter the comment via a free text or via the pre-defined text list.
  The acknowledgment and the entered comment are logged in the Audit Trail.

Logging tag value changes

Requirement

- "Configuration conforms to GMP" has been activated in the Runtime settings.

Procedure

1. Open the HMI tags editor and select the tag for which you want to make GMP settings.
2. Click "GMP relevant" under "Properties > Properties > "GMP" in the Inspector window.

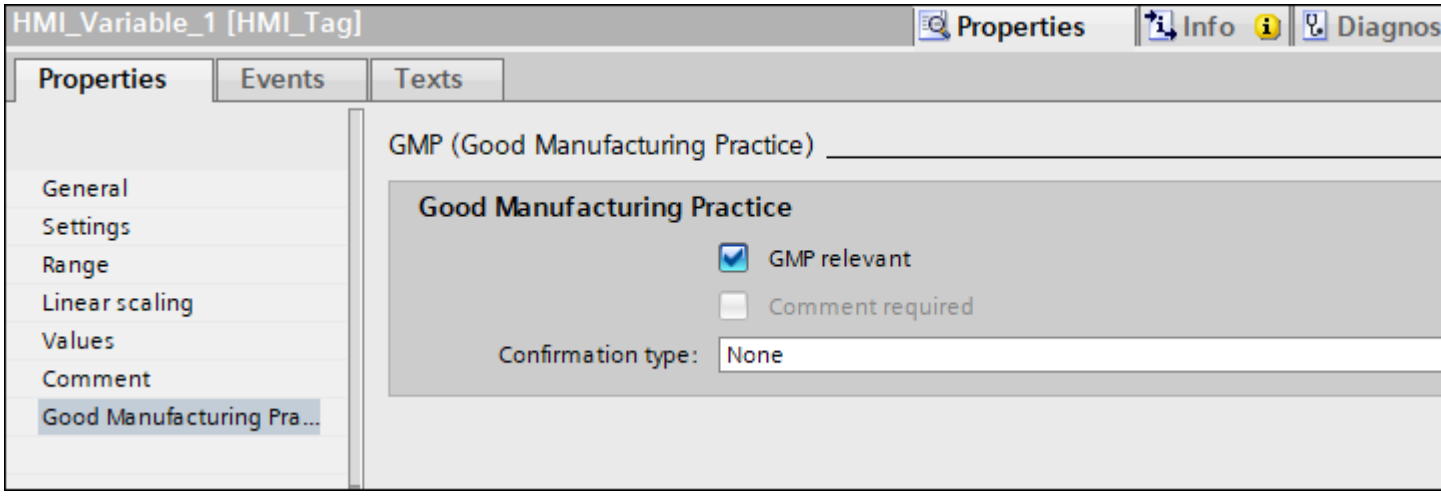


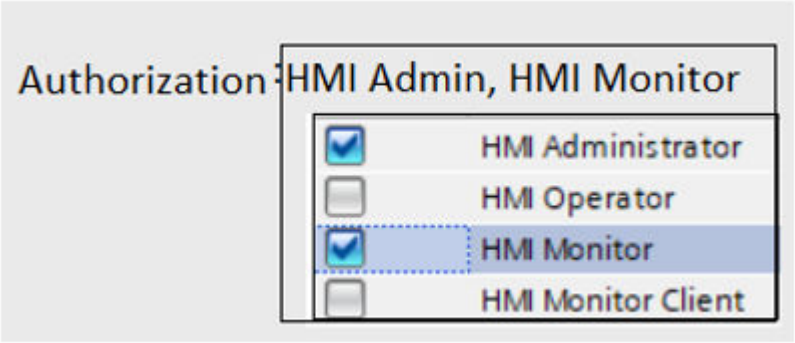
Figure 19-1 Marking a tag as GMP relevant

3. Specify how the user must confirm a value change in the "Confirmation type" selection field:
   - "None"
     If the value change is to be logged in the Audit Trail without user confirmation.
   - "Acknowledgment"
     If user acknowledgement of the value change is required.
4. Select the "Comment required" check box if the user is required to also enter a comment. This check box is only selected when "Acknowledgment" is specified under "Type of confirmation".

Confirmation type "Electronic signature"

If you have selected the electronic signature as the confirmation type, a drop-down list with the HMI roles is displayed.

Select which of the roles can confirm a value change.



Specifying comments via text lists

To keep the comments uniform, you can use text lists.

1. Create a text list with texts that can be selected for the comment.
2. In the runtime settings of the HMI device, select the text list in "Reasons for GMP-relevant tag".

19.1.3.2 Logging user actions

User actions with GMP-compliant configuration

Introduction

In a GMP-compliant configuration, user actions and system operations in runtime which are relevant for the quality of the process are recorded in an Audit Trail.

For example, a user logon to the system or the change of a tag value are saved in the log.

In runtime, user actions are saved in an Audit Trail under the following conditions:

- "Configuration conforms to GMP" has been enabled
- A user is logged on to the system

Logging modes

Configuration-dependent logging

The following processes are logged depending on the configuration of the tags of the project:

- Value changes of GMP-relevant tags by the user
  In addition to logging user actions, you can configure tags to require users to confirm or acknowledge specific actions and enter a comment on the change.

Manual logging by means of the "InsertElectronicRecord" system function

This system function is used to log actions in the Audit Trail that are not automatically logged in the Audit Trail.

Configuring the "InsertElectronicRecord" system function

Introduction

This system function is used to log user actions that are not entered automatically in the Audit Trail. Moreover, you can use this system and script function to request the user to enter an acknowledgment or a comment for the action.

You can configure this function even if the Unified Scada device is not configured as being GMP-compliant. However, current archiving in Runtime takes place only if the device is configured as GMP-compliant.

This function can be configured for screen objects, schedulers, global script. At present, this function is not available for faceplates.

In this example, the system function is assigned to a button. Every time the user operates this button, this action is logged in the Audit Trail.

Requirements

- "Configuration conforms to GMP" has been enabled.

Procedure

1. Click on a button in a screen.
2. Click on "Events" in the Inspector window.
3. In the function list, configure the "InsertElectronicRecord" system function to the "Click" event.

System function parameters

Parameter	Description	Type
Name	Specifies the name of the object.	String, HMI_tag, Screen object
Category	Specifies the name of the tag.	String, HMI_tag, Screen object
Operation type	Specifies the type of change.	1: New value 2: Modified value 3: Deleted value
Old value	Value before the change	Integer, Double, Bool, String, Color, HMI_tag, Screen object
New value	Value after the change	Integer, Double, Bool, String, Color, HMI_tag, Screen object

Parameter	Description	Type
Confirmation type	Specifies whether an acknowledgment is requested.	Specifies how the action must be confirmed. 0 = (None): No confirmation required, an entry is created in the Audit Trail 1 = (Acknowledgement): Acknowledgment, the user must acknowledge the action; an entry is created in the Audit Trail 2 = (Digital Signature): Electronic signature; a dialog window opens in which the user must enter the electronic signature - an entry is created in the Audit Trail
Reason (optional)	Is not supported	

Result

When a user operates the screen object in Runtime and triggers the event, the tag value is changed. An entry is created in the Audit Trail at the same time; depending on the parameters "Confirmation type" and "Reason", the entry is an acknowledgment and a comment.

GMP-compliant user administration

Central user management

Use the central user management to manage users and user groups centrally for multiple applications or HMI devices.

19.1.3.3 Recording system functions

Introduction

If system functions are triggered in runtime, this is recorded in the Audit Trail for some system functions. If specific system functions are used on a GMP-relevant object, the user must confirm the triggering.

Some system functions are not supported when using Audit. If you use these system functions in your project, you are solely responsible for them.

The following table shows which system functions are Audit-relevant and whether the user's signature is required:

System functions and Audit

Function (call in script)	Effect of /Audit
InsertElectronicRecord	Entered in Audit Trail
SetTagValue	Entered in Audit Trail
IncreaseTag	Entered in Audit Trail
DecreaseTag	Entered in Audit Trail

Function (call in script)	Effect of /Audit
ModifyBits	
SetBitInTag (SetBitInTag)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
ResetBitInTag (ResetBitInTag)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
InvertBitInTag (InvertBitInTag)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
SetBit (SetBit)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
ResetBit (ResetBit)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
InvertBit (InvertBit)	Entered in Audit Trail if the tag is GMP relevant Do not use the system function for tags that require acknowledgment or comment.
SetBitWhileKeyPressed (---)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
SetDataRecordToPLC (SetDataRecordToPLC)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
GetDataRecordTagsFromPLC (GetDataRecordFromPLC)	Entered in Audit Trail if the recipe is GMP relevant
ImportDataRecords (ImportDataRecords)	Entered in Audit Trail if the recipe is GMP relevant
ImportDataRecordsWithChecksum (ImportDataRecordsWithChecksum)	Entered in Audit Trail if the recipe is GMP relevant
ExportDataRecords (ExportDataRecords)	---
ExportDataRecordsWithChecksum (ExportDataRecordsWithChecksum)	---
LoadDataRecord (LoadDataRecord)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration

Function (call in script)	Effect of /Audit
SaveDataRecord (SaveDataRecord)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
SetDataRecordTagsToPLC (SetDataRecordTagsToPLC)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
GetDataRecordTagsFromPLC (GetDataRecordTagsFromPLC)	Entered in Audit Trail if the recipe is GMP relevant
SetRecipeTags (SetRecipeTags)	Entered in Audit Trail if the recipe is GMP relevant
GetDataRecordName (GetDataRecordName)	---
ClearDataRecordMemory (ClearDataRecordMemory)	Not supported
ClearDataRecord (ClearDataRecord)	Entered in Audit Trail if the recipe is GMP relevant
PrintScreen (PrintScreen)	---
PrintReport (PrintReport)	---
RecipeViewSaveDataRecord (---)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
RecipeViewSaveAsDataRecord (---)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
RecipeViewNewDataRecord (---)	---
RecipeViewClearDataRecord (---)	Entered in Audit Trail if the recipe is GMP relevant
RecipeViewGetDataRecordFromPLC (---)	Entered in Audit Trail if the recipe is GMP relevant
RecipeViewSetDataRecordToPLC (---)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
RecipeViewSynchronizeDataRecordWithTags (---)	Entered in Audit Trail if the recipe is GMP relevant Confirmation is dependent on recipe configuration
RecipeViewRenameDataRecord (---)	Entered in Audit Trail if the recipe is GMP relevant
RecipeViewBack (---)	---
RecipeViewOpen (---)	---
RecipeViewMenu (---)	---
TrendViewScrollForward (---)	---
TrendViewScrollBack (---)	---
TrendViewExtend (---)	---

Function (call in script)	Effect of /Audit
TrendViewCompress (---)	---
TrendViewBackToBeginning (---)	---
TrendViewStartStop (---)	---
TrendViewSetRulerMode (---)	---
TrendViewBackToBeginning (---)	---
StatusForceGetValues (---)	Not supported
StatusForceSetValues (---)	Not supported
AlarmViewAcknowledgeAlarm (---)	Entered in
AlarmViewEditAlarm (---)	---
AlarmViewShowOperatorNotes (---)	---
HTMLBrowserBack (---)	Not supported
HTMLBrowserForward (---)	Not supported
HTMLBrowserRefresh (---)	Not supported
HTMLBrowserStop (---)	Not supported
ScreenObjectCursorUp (---)	---
ScreenObjectCursorDown (---)	---
ScreenObjectPageUp (---)	---
ScreenObjectPageDown (---)	---
PressButton (---)	---
ReleaseButton (---)	---
SmartClientViewConnect (---)	Not supported
SmartClientViewDisconnect (---)	Not supported
SmartClientViewReadOnlyOn (---)	Not supported
SmartClientViewReadOnlyOff (---)	Not supported
SmartClientViewRefresh (---)	Not supported
SmartClientViewLeave (---)	Not supported
ShowAlarmWindow (ShowAlarmWindow)	---
ClearAlarmBuffer (ClearAlarmBuffer)	---
ShowSystemAlarm (ShowSystemAlarm)	---
SetAlarmReportMode (SetAlarmReportMode)	---
Logoff (Logoff)	Entered in Audit Trail
GetPassword (GetPassword)	---
GetGroupNumber (GetGroupNumber)	---
ExportImportUserAdministration (ExportImportUserAdministration)	Import of user administration is entered in Audit Trail Export is not entered in Audit Trail
Logon (Logon)	Entered in Audit Trail
GetUserName (GetUserName)	---
TraceUserChange (---)	---
ShowLogOnDialog (---)	---

Function (call in script)	Effect of /Audit
LinearScaling (LinearScaling)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
InverseLinearScaling (InverseLinearScaling)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
IncreaseFocusedValue (---)	---
DecreaseFocusedValue (---)	---
OpenCommandPrompt (OpenCommandPrompt)	Not supported
OpenControlPanel (OpenControlPanel)	Not supported
ActivateCleanScreen (---)	---
AdjustContrast (---)	---
CalibrateTouchScreen (CalibrateTouchScreen)	---
OpenScreenKeyboard (OpenScreenKeyboard)	---
OpenTaskManager (OpenTaskManager)	Not supported
BackupRAMFileSystem (BackupRAMFileSystem)	Not supported
SetAcousticSignal (SetAcousticSignal)	---
ShowOperatorNotes (ShowOperatorNotes)	---
AcknowledgeAlarm (AcknowledgeAlarm)	Entered in Audit Trail
GoToHome (GoToHome)	---
GoToEnd (GoToEnd)	---
EditAlarm (EditAlarm)	---
DirectKeyScreenNumber (---)	Not supported
DirectKey (---)	Not supported
SetDeviceMode (SetDeviceMode)	Entered in Audit Trail
SetDisplayMode (SetDisplayMode)	---
SetConnectionMode (SetConnectionMode)	Entered in Audit Trail
SetScreenKeyboardMode (SetScreenKeyboardMode)	---
ChangeConnection (ChangeConnection)	Not supported
SetLanguage (SetLanguage)	---
SetWebAccess (---)	Not supported
StartProgram (StartProgram)	Not supported
ShowSoftwareVersion (ShowSoftwareVersion)	---
SimulateTag (---)	Not supported
StopRuntime (StopRuntime)	Entered in Audit Trail
ControlWebServer (ControlWebServer)	Not supported
ControlSmartServer (ControlSmartServer)	Not supported
OpenInternetExplorer (OpenInternetExplorer)	---
SendEMail (SendEMail)	---
UpdateTag (---)	---
ClearAlarmBufferProTool (ClearAlarmBufferProtoolLegacy)	Not supported

Function (call in script)	Effect of /Audit
Encoding(Encode)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration
EncodeEx(Encode)	Entered in Audit Trail if the tag is GMP relevant Confirmation is dependent on tag configuration

19.1.3.4 Standard entries in the Audit Trail

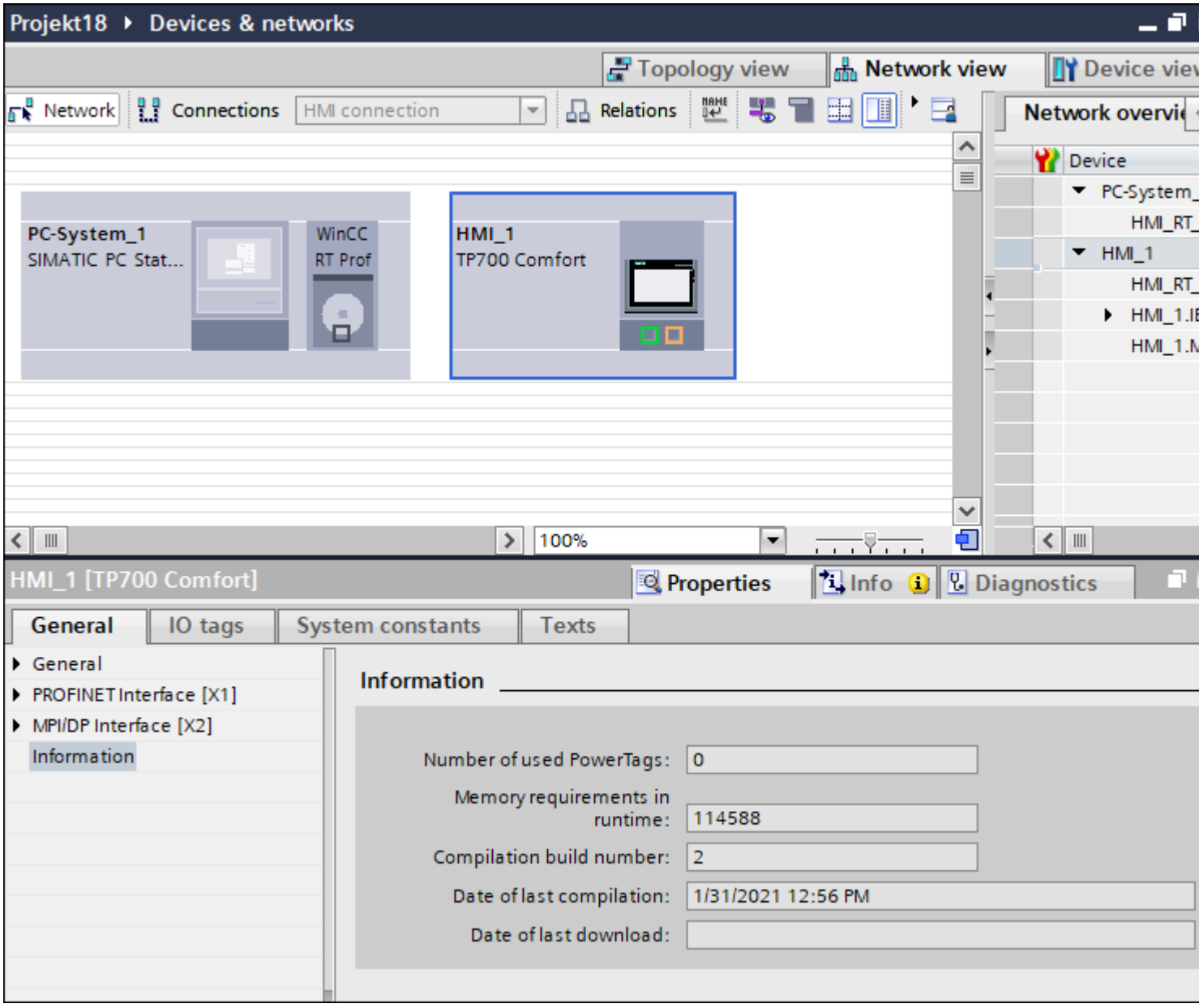
General entries in the Audit Trail

Individual operator actions are overwritten during logging. The following table shows examples of such entries:

Log entry	Description
Change of the tag value 'HMI_Tag_1' from '0' to'55'	Changing the value of a tag
User logged off.	Logging out a user.
Shutting down application.	Exiting the program.
Runtime start of WinCC Runtime Advanced	Starting WinCC Runtime Advanced
'Project.HMI_1 - 0' Build 2.	Information on the project, the device and the current version number

Version number of the device

The version number is incremented with each compilation of the device. You can find the version number in the Inspector window under "Properties > Properties > Information".



19.2 Creating production reports

19.2.1 Basics

19.2.1.1 Introduction

Note

Restriction for Unified Comfort Panel

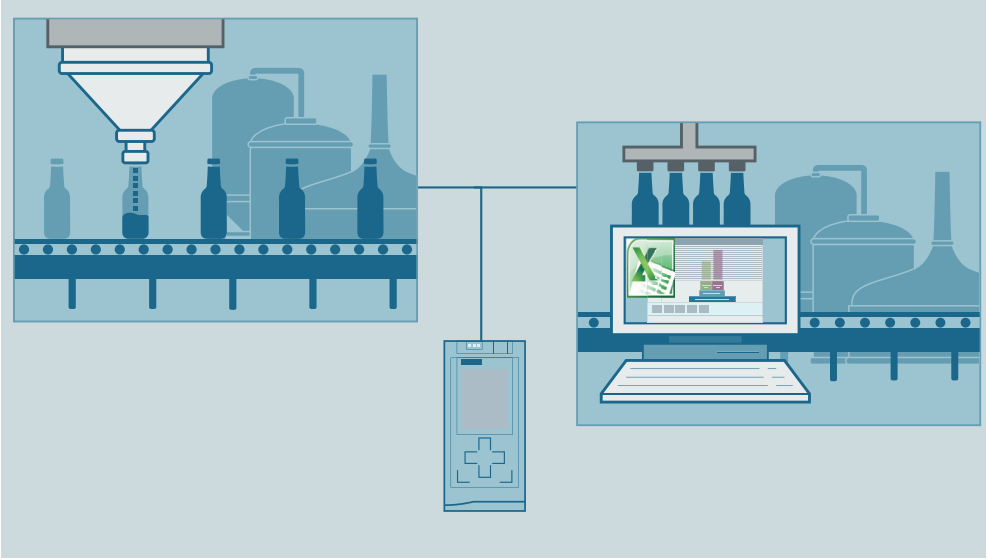
Contexts are not supported in V17 for Unified Comfort Panel. This option is not available in a report template with a Unified Comfort Panel as data source. When you generate a report on a Unified Comfort Panel whose report template uses this option, error entries are generated in the "ErrorLog" worksheet of the report.

Introduction

With WinCC Unified Reporting, you can generate tabular production reports in Runtime for the following project data:

- Logging tags and tags
- Log alarms
  Log the alarm properties of the recorded alarms and statistical calculations for display duration and frequency of the recorded alarms.
- Contexts:
  Log which contexts were running during a specific period or which project data occurred during the runtime of a specific context.
  - User-defined contexts:
    These contexts are created and executed by a program created with the ODK API.
  - System-generated contexts
    When the Performance Insight and Calendar option packages are installed, these contexts are executed by the system during Runtime.
- Audit Trail of the Runtime device
- If Plant Intelligence options are installed, you can use the WinCC Unified Local Reporting option to generate production reports for additional project data.
  You can find more information in the Help for the respective Plant Intelligence option.

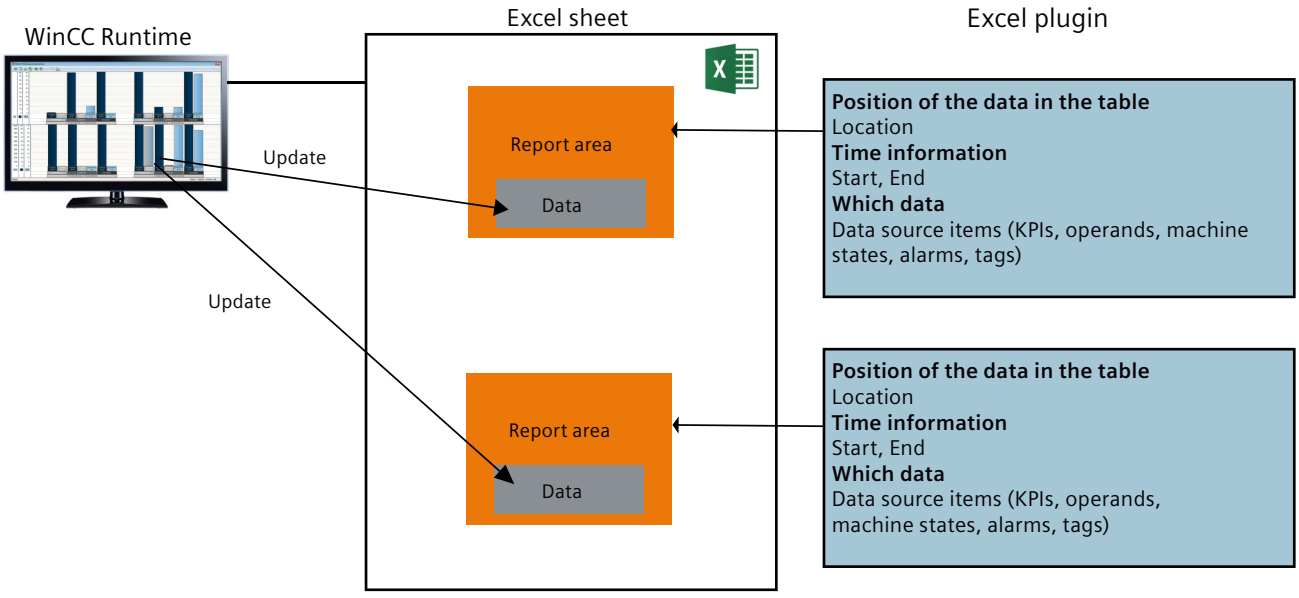
The production reports can be generated as XLSX file or PDF file and sent automatically as an e-mail to a specified group of recipients. For example, you can generate an XLSX report that outputs all alarms occurring in a production line. You then distribute or archive the report for analysis purposes.



Functional scope in the add-in

You create the report templates in an Excel add-in. For this purpose, Reporting offers the following functions in the add-in:

- Selecting a data source:
  - Online: A connection to a Runtime server on which a Runtime project is running.
  - Offline: A configuration file that contains the data of a Runtime project
- Definition of single value segments and time series segments
- Selection of the data source items of the segments
  Possible data source items:
  - Logging tags and tags
  - Log alarms
  - Contexts
- Definition of the report period (absolute or relative)
- Creation of type-specific data source item configurations
- Execution of individual segments or all segments for test purposes



Functional scope in Runtime

In the "Reports" control in Runtime, you configure report jobs that use the report templates defined in the Excel add-in. To do so, Reporting offers the following functions in Runtime:

- Maintenance of the global email settings (contact data and SMTP server configuration)
- Maintenance of job parameters, especially import and export of report templates
- Creating new report jobs and managing existing report jobs
- Overview of the generated reports
- Download or deletion of the reports

19.2.1.2 Basics of Reporting

Report templates

A *report template* is an Excel file (.xslx) that was created with the WinCC Unified Excel add-in. The report template has access to the data of the data source with which the add-in is connected.

For each report template, you define which segments are contained in the reports using the template and which data source items are evaluated by the segments.

After you have imported report templates into the "Reports" control in Runtime, you can select them for configuring report jobs.

Data sources

The *data source* is the source from which you select data source items when you configure the report template.

The following connection modes and data sources are available:

- Connection mode: Online
  The data source is the project that is running on the Runtime server to which the add-in is connected.
- Connection mode: Offline
  Data source is a configuration file. You generate the configuration file by exporting the data source items of the project to a file in the "Reports" control in Runtime. You can use this file to create additional report templates without connecting to a runtime server.

Options and data source items

Options control the types of data source items to which the report template has access.

Data source items are the specific objects whose data is read from the Runtime project during report generation.

The following options and types of data source items are available in Reporting, depending on the installed software:

Software	Option	Types of data source items
WinCC Unified basic installation	Alarms	Logging alarms Alarm statistics for logging alarms
WinCC Unified basic installation	Logging tag	Logging tags
WinCC Unified basic installation	Tag	Tags
WinCC Unified basic installation	User-defined column	User-defined texts or Excel formulas
WinCC Unified basic installation	Context	User-defined contexts Not available for Unified Comfort Panel
WinCC Unified basic installation	Audit	Audit
Performance Insight option package	Performance Insight	Local KPIs and operands of the PI option Performance Insight: • KPIs • Logged KPIs • Operands (counters and numerical operands) • Machine states • System-generated contexts
Line Coordination option package	Line Coordination	Jobs
Calendar option package	Context	System-generated contexts

Report jobs and job parameters

A *report job* is a job for generating reports in Runtime. A new report is generated each time the report job is performed.

The *job parameters* of the report order determine the details of its execution, such as which trigger it has, which report template it uses and the format of the report.

Report jobs are executed automatically when their trigger event occurs or manually by the user.

Reports

A *report* (production report) is an XLSX file or PDF file that is generated when a report job is executed in Runtime. The data source items from the Runtime project defined in the report template are read during generation, and their data are imported into a table in the report.

Using general Excel functions

In addition to the specific add-in functions, you also have access to the standard Excel functions in a report template. These include:

- Layout functions
- Functions for graphical preparation or analysis of the data imported from Runtime, such as charts, pivot tables and formulas

See also Tips on design and layout (Page 7717).

19.2.1.3 General requirements and restrictions

Installing the Excel add-in

The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

Update the operating system and the Excel version if necessary.

To install the add-in with a local Excel installation, MS Excel with build 16.0.6769 or higher is required.

Note

Note the Microsoft upgrade restrictions

If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

IIS settings for standalone installation of the Excel Add-In

To install the Excel Add-In on a PC without Unified Runtime, the same IIS (Internet Information Services) settings must be active in Windows that are required to install WinCC Unified Runtime on a PC.

You can find additional information in the "SIMATIC Unified PC Installation" user help section on the software and hardware requirements.

Unified Comfort Panel

The following restrictions apply to generating reports on Unified Comfort Panels:

Contexts	Contexts are not supported in V17 for Unified Comfort Panel. This option is not available in a report template with a Unified Comfort Panel as data source. When you generate a report on a Unified Comfort Panel whose report template uses this option, error entries are generated in the "ErrorLog" worksheet of the report.
Storage location of the Reporting database	The following folder on the SD card plugged into the panel is preconfigured as the storage location of the reporting database: • Device version V18: `media/simatic/X51/Reports` You can configure and load another storage location in TIA Portal in the Runtime settings of the Panel. • Device version <V18: `media/simatic/X51`
Storage location for reports	The "Reports" folder on the SD card inserted in the Panel is permanently pre-configured as the local main storage location for reports: `media/simatic/X51/Reports` For Panels with device version V18, you can configure and load a different local main storage location in the Runtime settings of the Panel in TIA Portal.

Enable Reporting

The use of Reporting requires that the Reporting functionality was enabled:

- For configuring report templates:
  Reporting must be enabled for the Runtime project that serves as the data source.
- For configuring report jobs and generating reports in Runtime as well as in a simulation:
  Reporting must be enabled for the Runtime project that is running on the HMI device or is being simulated.

The reporting functionality of a Runtime project is enabled in TIA Portal in the Runtime settings of its HMI device with the option "Enable reporting".

Note

Devices with a device version lower than V18

Reporting is always enabled for HMI devices with a device version lower than V18.

See also

Version compatibility (Page 7651)

Configuring Reporting-specific Runtime settings (Page 7655)

19.2.1.4 Version compatibility

Introduction

When loading a Runtime project for which the "Reports" control has been configured, the general rules for version compatibility of WinCC Unified apply.

The rules described here also apply for the interaction between add-in, data source, report template and runtime version of the project in which reports are generated.

Compatibility between add-in and data source

The add-in can use the following data sources:

Add-in	Online data source	Offline data source
V16	Runtime project V16	Configuration file generated with a Runtime project V16
V17	Runtime project V16 or V17	Configuration file generated with a Runtime project V16 or V17

Compatibility between add-in and report template

The following report templates can be opened and edited in the add-in:

Add-in	Report template
V16	Created with a V16 add-in
V17	• Created with a V17 add-in • Created with a V16 add-in If the add-in is connected to a V17 data source when you open the report template, you will be prompted to migrate the report template to V17. If the add-in is connected to a V16 data source when the report template is opened, no migration is necessary.

Note

Migration of report templates

The migration of the report template is not reversible. A report template migrated from V16 to V17 can no longer be opened in a V16 add-in.

If migration is not desired, connect the add-in to a V16 data source before opening the report template.

Note

Scope of functions of report templates

The functions available in the configuration of the report template in the add-in depend on the version of the data source used by the add-in.

Compatibility between report template and runtime project

In a runtime project, reports can be generated using the following report templates:

Report template	Version of the runtime project
V16	V16 and V17
V17	V17

See also

Basics on version compatibility (Page 187)

19.2.2 Complete workflow for using production reports

Introduction

The use of reports (production reports) requires preliminary work in the Engineering System, in the Excel add-in and in Runtime. The exact workflow depends on where the report template for generating the report is used:

- Internally in the project
  A report template is based on Project A. It is used in Runtime in Project A to generate reports.
- Across projects
  A report template is based on Project A. It is used in Runtime in Project B to generate reports. Project A and Project B were downloaded into the same Runtime.
  The data source items added to the report template must be findable in both projects. Make sure that the names are consistent or that you have a uniform plant model.
- Across Runtimes
  A report template is based on Project A. It is used in Project B to generate reports. Project A and Project B were downloaded into different Runtimes.
  In this case, the following applies:
  - The data source items added to the report template must be findable in both projects. Make sure that the names are consistent or that you have a uniform plant model.
  - In both Runtimes, the same options that are used in the report template must be installed.

Workflow for cross-project use of report templates in the same Runtime

1. In the Engineering System, configure the HMI device that serves as data source for the Excel add-in during configuration of the report templates.
   - Configure alarms and tags.
   - When reports are to evaluate context data, configure the plant model as well as other objects and settings that are necessary to start system-generated contexts.
     Example: To evaluate shift contexts of the Calendar PI option, you configure the plant model, the time model, a calendar and a calendar control.
   - Enable Reporting in the Runtime settings of the device.
2. Select one of the following options:
   - To create the report template before the HMI device goes live, start a simulation for the HMI device.
     Requirement: The engineering system and Runtime are installed on the same device.
     The device is downloaded to Runtime. Its project (project A) is started in simulation mode.
   - To create the report template during productive operation of the HMI device, load the device from the Engineering System into the Runtime and start its project (project A).

   Note

   Evaluation of context-relevant data

   To ensure that reports generated in runtime evaluate context-relevant data, contexts must be executed in the project during runtime.

   The system-generated contexts of the PI options Performance Insight and Calendar are started automatically while a project is running in Runtime.

   You can also use the ODK API to write a program that defines, starts and stops user-defined contexts for a project.

3. (Optional) To work with an offline connection in the Excel add-in, export an offline configuration file in the "Reports" control in Runtime.
4. Start the Excel add-in and set up the data source.
   Select one of the following options:
   - Online connection: Select the Runtime server onto which you loaded the HMI device.
   - Offline connection: Load the offline configuration file created in Step 3 as the data source.
5. Configure a report template in the Excel add-in.
6. (Optional) Test a report template by running it in the Excel add-in and reading the project data into Excel.

7. In the Engineering System, configure the HMI device for which you want to generate reports in Runtime:
   - Place the "Reports" control in one of the screens of the HMI device.
   - Configure alarms, tags and the plant model if the reports are to evaluate context data. Make sure that this data matches the data of the HMI device from step 1.
   - Enable Reporting in the Runtime settings of the device.
   - (Optional) In the Runtime settings of the device, configure the storage location for reports and the storage location of the reporting database.
8. Load the HMI device from the engineering system into Runtime and start its project (Project B).
9. In Runtime, go to the screen with the "Reports" control.
10. (Optional) To send emails when reports are created, configure the global email settings.
11. Configure the job parameters.
    To do this, import the report template defined in step 4, among others.
12. Configure a report job that uses the report template.
13. (Optional) Perform report orders manually.
14. In the control, get an overview of which reports have been generated.
15. Download the reports, if necessary.
16. (Optional) To reuse the configuration of the "Reports" control, such as on a device in another network, transfer the existing configuration from the control from one device to the control of the other device.

See also

Reporting (Page 7145)

Reporting (Page 7180)

Exporting an offline configuration file (Page 7743)

Setting up a data source (Page 7663)

Configuring report templates (Page 7668)

Configuring production reports in the engineering system (Page 7655)

Setting global email settings (Page 7725)

Configuring task parameters (Page 7727)

Configuring report tasks (Page 7734)

Running a report job manually (Page 7742)

Downloading reports (Page 7742)

Transferring the control configuration (Page 7744)

Workflow for working with reports in Runtime (Page 7719)

19.2.3 Configuring production reports in the engineering system

19.2.3.1 Configuring Reporting-specific Runtime settings

Procedure

1. Open "Runtime settings > Reporting" in the project tree below the HMI device.
2. Select the "Enable Runtime" option.
   When the option is disabled, it is not possible to use the Runtime project as data source for report templates or configure report jobs in Runtime and generate reports.
3. (Optional) Configure the storage location of the reporting database.
4. (Optional) Configure the storage location for the reports generated in Runtime.

You can find more information on configuring these storage locations for Unified Comfort Panel here (Page 7145), for Unified PC here (Page 7180).

19.2.3.2 Inserting a "Reporting" control in a screen

Procedure

1. Select the HMI device on the "Devices" tab.
2. Open the "Screens" folder.
3. Open the screen.
4. In the "My controls" pane, select the "Reporting" control and place it on the screen.

19.2.4 Creating report templates for production reports

19.2.4.1 Requirements

General requirements and restrictions

Installing the Excel add-in

The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

Update the operating system and the Excel version if necessary.

To install the add-in with a local Excel installation, MS Excel with build 16.0.6769 or higher is required.

Note

Note the Microsoft upgrade restrictions

If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

IIS settings for standalone installation of the Excel Add-In

To install the Excel Add-In on a PC without Unified Runtime, the same IIS (Internet Information Services) settings must be active in Windows that are required to install WinCC Unified Runtime on a PC.

You can find additional information in the "SIMATIC Unified PC Installation" user help section on the software and hardware requirements.

Unified Comfort Panel

The following restrictions apply to generating reports on Unified Comfort Panels:

Contexts	Contexts are not supported in V17 for Unified Comfort Panel. This option is not available in a report template with a Unified Comfort Panel as data source. When you generate a report on a Unified Comfort Panel whose report template uses this option, error entries are generated in the "ErrorLog" worksheet of the report.
Storage location of the Reporting database	The following folder on the SD card plugged into the panel is preconfigured as the storage location of the reporting database: • Device version V18: `media/simatic/X51/Reports` You can configure and load another storage location in TIA Portal in the Runtime settings of the Panel. • Device version <V18: `media/simatic/X51`
Storage location for reports	The "Reports" folder on the SD card inserted in the Panel is permanently pre-configured as the local main storage location for reports: `media/simatic/X51/Reports` For Panels with device version V18, you can configure and load a different local main storage location in the Runtime settings of the Panel in TIA Portal.

Enable Reporting

The use of Reporting requires that the Reporting functionality was enabled:

- For configuring report templates:
  Reporting must be enabled for the Runtime project that serves as the data source.
- For configuring report jobs and generating reports in Runtime as well as in a simulation:
  Reporting must be enabled for the Runtime project that is running on the HMI device or is being simulated.

The reporting functionality of a Runtime project is enabled in TIA Portal in the Runtime settings of its HMI device with the option "Enable reporting".

Note

Devices with a device version lower than V18

Reporting is always enabled for HMI devices with a device version lower than V18.

See also

Version compatibility (Page 7651)

Installation of the Reporting add-in

Note

Regular updates of operating system and MS Excel

The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

Update the operating system and the Excel version if necessary.

To install the add-in with a local Excel installation, MS Excel with build 16.0.6769 or higher is required.

Note

Note the Microsoft upgrade restrictions

If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

Procedure

1. Install the Excel manifest on the computer.
2. Set up read access to the installation path of the Excel manifest.
3. Add the add-in to Excel.

See also

Installing the Excel manifest (Page 7658)

Setting up read access to the Excel manifest (Page 7658)

Adding the Reporting add-in in Excel (Page 7659)

MS help for autoloop (https://docs.microsoft.com/en-us/office/troubleshoot/error-messages/cannot-open-add-in-from-localhost)

Installing the Excel manifest

Procedure

1. In the installation package of WinCC Unified on "DVD_2", double-click the file "Support\Reporting\SIMATIC_WinCC_Unified_Reporting_<Version number>.exe".
2. Select the target directory to which the underlying ZIP file is extracted and confirm your input. The ZIP file is extracted and setup starts automatically.

   Note

   Start setup manually

   To start the setup manually after the file was extracted, select the option "Extract the setup files without being installed".

   Start the setup later by running the "Setup.exe" file as administrator in the target directory.

3. Follow the setup instructions.
4. In the "Configuration" step, select the option for the Excel add-in.
5. Click "Next" and follow the setup instructions.

See also

Installation of the Reporting add-in (Page 7657)

Setting up read access to the Excel manifest

Requirement

The Excel manifest is installed.

Procedure

Give the users that create templates with the Excel add-in read access to the installation path of the Excel manifest: <target directory>\WinCCUnifiedReporting\Excelmanifest

Note

This step is also necessary if the user belongs to a group in the user management with general read permission.

See also

Installing the Excel manifest (Page 7658)

Installation of the Reporting add-in (Page 7657)

Adding the Reporting add-in in Excel

Requirement

- The Excel manifest is installed on the PC.
- Read access to the installation path of the Excel manifest is set up.
- The following software is available on the computer:
  - Local Excel
    MS Excel (Build 16.0.6769 or higher)

    Note

    Regular updates of operating system and MS Excel

    The installation of the Reporting add-in on a computer requires that the operating system and the local MS Excel installation are regularly updated.

    If there are problems with the installation, check the version of the local MS Excel installation. Lengthy maintenance intervals between the operating system and Excel can cause problems during installation of the add-in.

    Update the operating system and the Excel version if necessary.

    Note

    Note the Microsoft upgrade restrictions

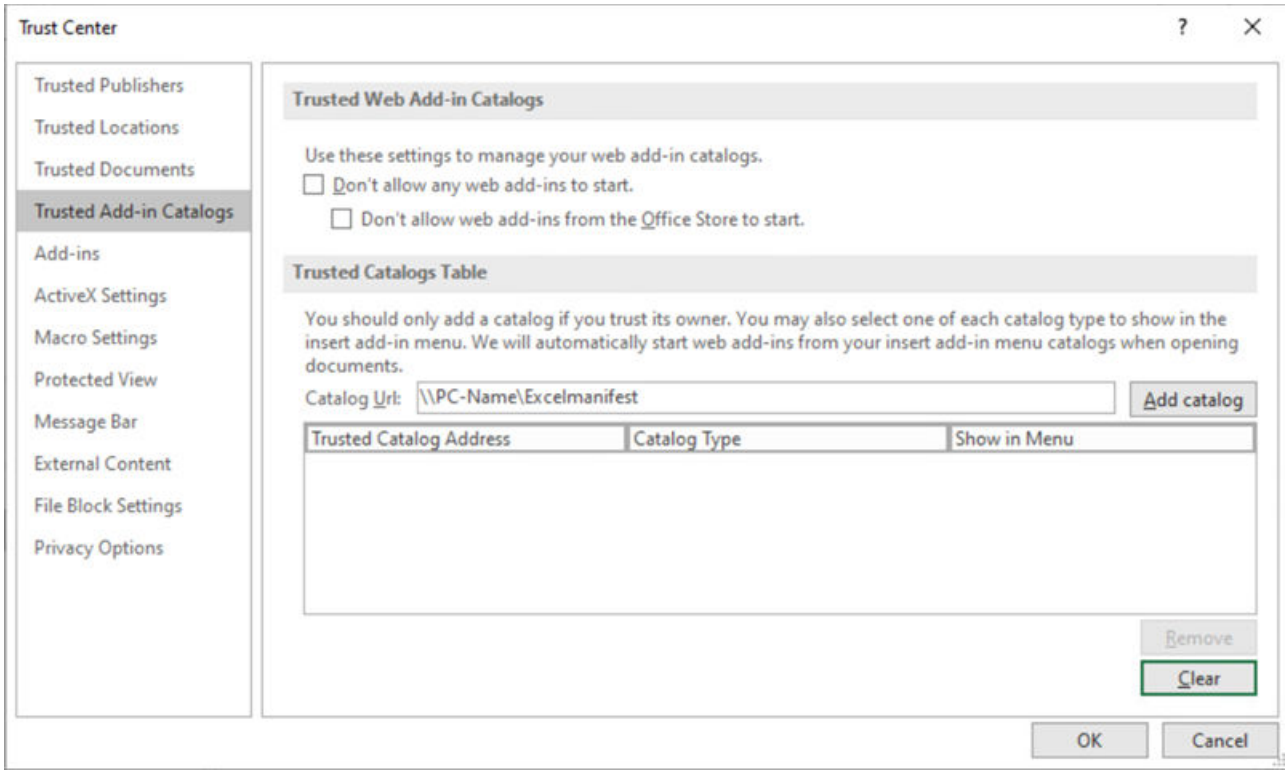
    If you have an Excel installation that cannot be upgraded to Build 16.0.6769 or higher (for example, because Excel was installed using a single Office license), purchase a current Office version or use Online Office.

  - Or Office online

Procedure

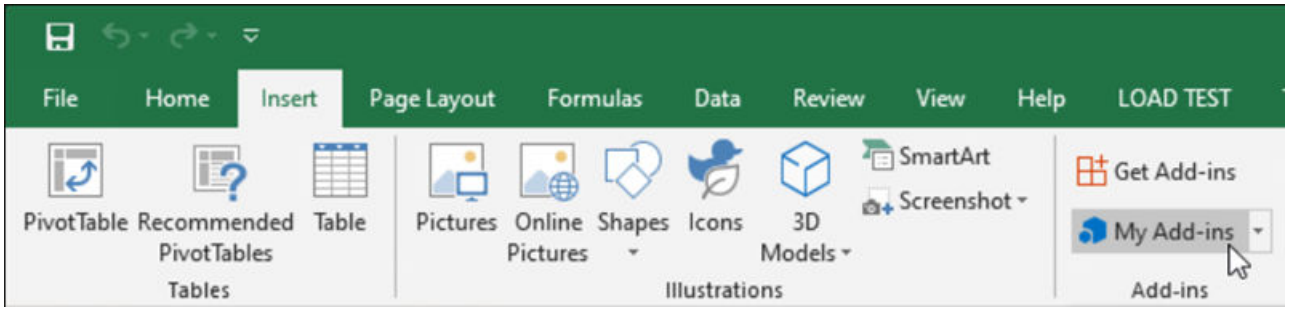
1. Open Microsoft Excel.
2. Open the "Trust Center" under "File" > "Options".
3. Click "Trust Center Settings".
4. Click "Catalogs of trusted add-ins".

5. Add the catalog using the URL "\\<Computer name>\Excelmanifest".



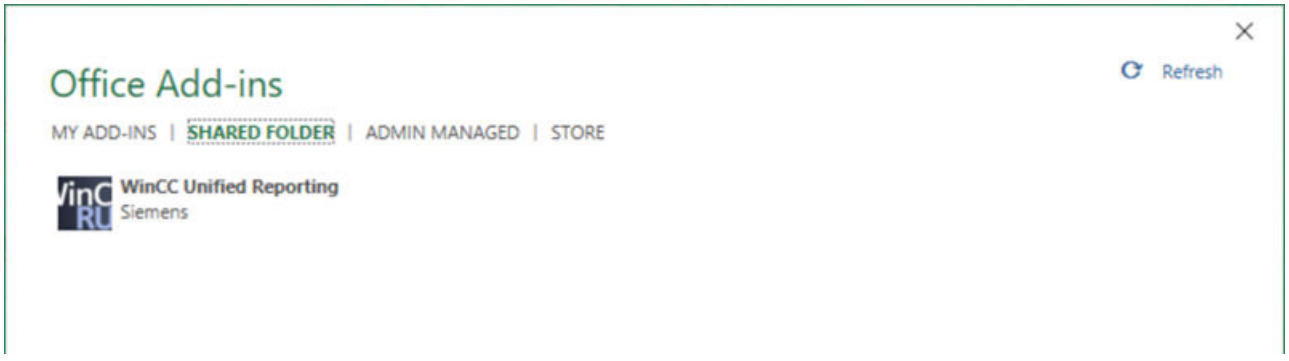
6. Make sure that the check mark in the "Show in Menu" column is set.
7. End and restart Excel.

8. In the "Insert" menu, click "My Add-ins".



In the "Office Add-ins" dialog box, the Siemens add-in is displayed under "Shared folders".

9. Select the add-in and click "Add".



See also

Installing the Excel manifest (Page 7658)

Setting up read access to the Excel manifest (Page 7658)

Installation of the Reporting add-in (Page 7657)

Configuring Internet Explorer and Edge

The Reporting Excel add-in uses the certificate that was selected during installation of WinCC Unified Runtime or later in "WinCC Unified Configuration".

Some browsers do not consider self-signed certificates as trusted. If you use a self-signed certificate for WinCC Unified Runtime, you must add the certificate to the list of trusted certificates in Internet Explorer or Edge on the device on which the Excel add-in is installed.

For detailed information on handling certificates, see the Runtime Readme.

Procedure

The following section describes the procedure for adding a self-signed certificate to the list of trusted certificates, using Internet Explorer as an example:

1. Start Internet Explorer.
2. In the address line, enter the host name or the IP when creating the certificate.
   You will receive a security warning.
3. Click "Continue to this website (not recommended)".
4. Click "Install certificate".
5. Click "Place all certificates in the following store" and "Browse".
6. Click "Trusted Root Certification Authorities" followed by "OK".

   Note

   Do not use the preset options for automatic selection of the certificate store.

7. Exit the dialog.
8. If you receive a security warning as to whether you want to trust the certificate, confirm it with "Yes".
9. Load the page again.

See also

MicrosoftHelp_IE_ZertifikatInstallieren (https://medium.com/@ali.dev/how-to-trust-any-self-signed-ssl-certificate-in-ie11-and-edge-fa7b416cac68)

19.2.4.2 Login

A login dialog opens in the Excel add-in in the following cases:

- After start of Excel and the add-in
- When using an online connection: When the connection to the Runtime server must be re-established.
  Examples:
  - Runtime has been reloaded.
  - The security token has expired due to a timeout.

Requirement

- The add-in is installed.
- When using an online connection:
  - A Runtime server is accessible.
  - A Runtime project is running on the server.

Procedure

In order to use an online connection, log onto a Runtime server:

1. Under "Server", enter the name of the server on which the project that is to serve as data source for the report template is running.
   Use the same spelling as when the Runtime server certificate was created.

   Note

   If Runtime is installed on the same computer as the add-in, use of the name "localhost" is not permitted.

2. Enter the user name and password of a user that is registered on the server in the Runtime user management.
3. Click "Login".

In order to use an offline connection, click "Go offline".

Result

Online connection

The add-in is connected to the Runtime server and the options available there are loaded.

You can now create report templates.

Offline connection

Before you create report templates, set up the offline connection.

See also

Setting up an offline connection (Page 7666)

Installation of the Reporting add-in (Page 7657)

19.2.4.3 Setting up a data source

Using an online connection

When an online connection is present, the add-in establishes a connection to a Runtime server. The project running on the server serves as data source for the add-in.

The connection settings allow you to:

- Change the connected Runtime server to another Runtime server
- When a report template that was created with a different Runtime server than the currently connected server is reused: check the options available on the server and delete the options that were not loaded

Setting up an online connection

Requirements

- A Runtime server is accessible.
- A Runtime project is running on the server.

Procedure

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Click "Online" under "Connections" in the add-in.
3. Under "Server", enter the server name.
   Use the same spelling as when the Runtime server certificate was created.

   Note

   If Runtime is installed on the same computer as the add-in, use of the name "localhost" is not permitted.

4. Click "Load".

Result

- A server node is created.
- The add-in is connected to the Runtime server and its options are loaded.
  Data source items of these options can be added to report templates. Their data can be read in from Runtime to Excel.

  Note

  To check which options were loaded, click on the server node.

  Options that are being used in the currently open report template but are not available on the connected server have a red icon. You can remove the option:

- If no connection can be established or an incorrect server name has been entered, the add-in will display a corresponding error message.

See also

Removing options (Page 7664)

Removing options

Introduction

If you reuse report templates across servers, e.g. in order to adapt an existing template for another project, it may be necessary to remove unavailable options from the connection settings.

The procedure for this is presented using the Performance Insight option as an example.

Requirement

- The add-in was connected to a server on which the Performance Insight (PI) option is installed.
- A report template that uses KPIs was created with the add-in.
- The add-in was then connected to a server without the Performance Insight option installed for the purpose of adapting the template to the project running there.

Removing an option

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Under "Connections", click on "Online".
3. Select the server node.
   You see the loaded options under the server node:

	Available options The following applies to data source items of these options: • They can be added to the report template. • Their data can be read in from Runtime to Excel in the add-in.
	Unavailable options In the example: Performance Insight The following applies to data source items of these options: • They cannot be added to the report template. • If the report template already has a data source element of this option, its data cannot be read in from Runtime to Excel.

4. Select the "Performance Insight" option under the server node.
5. Click the "Delete" button next to the option.
6. Confirm your input.

Result

The option is removed from the connection settings.

Next, remove all data source items of this option from the report template.

Reloading an option

When the add-in is connected to a Runtime server, all options available on the server are loaded.

To reload an option that was deleted in the connection settings but is available on the server, select the server node and click "Load".

Using an offline connection

With the offline connection, the add-in uses a configuration file as data source.

The connection settings allow you to:

- Change the configuration file used
- When reusing a report template with a configuration based on a Runtime server different to that of the currently selected configuration file: Check the available options and delete the options that were not loaded.

Setting up an offline connection

Requirement

An offline configuration file was created in the "Reports" control in Runtime. The configuration file is available on the device.

Procedure

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Under "Connections", click on "Offline".
3. Click "Open offline configuration".
4. Select the desired file in the window that opens and confirm your entries.
5. Click "Load".
6. Select the desired options.
7. Confirm your entries.

Result

- A server node is created. The node bears the name of the server on which the configuration file is based.
- The configuration file, together with its options, is loaded into the add-in. The data of the configuration file is available for configuring the report template.

Note

To check which options were loaded, click on the server node.

Options that are being used in the currently open report template but are not available in the configuration file have a red icon. You can remove the option:

See also

Removing options (Page 7667)

Exporting an offline configuration file (Page 7743)

Removing options

Introduction

If you reuse report templates across servers, e.g. in order to adapt an existing template for another project, it may be necessary to remove unavailable options from the connection settings.

The procedure for this is presented using the Performance Insight option as an example.

Requirement

- The add-in was changed over to an offline connection whose configuration file does not include Performance Insight.
- A report template was opened in the add-in whose configuration is based on a connection to a Runtime server on which Performance Insight is installed.

Removing an option

1. In the "Data sources" group on the "WinCC Unified" tab, click on "Connections".
2. Under "Connections", click on "Offline".
3. Select the server node.
   You see the loaded options under the server node:

	Available options The following applies to data source items of these options: • They can be added to report templates. • Their data can be read in from the configuration file to Excel.
	Unavailable options In the example: Performance Insight The following applies to data source items of these options: • They cannot be added to the report template. • If the report template already has a data source element of this option, its data cannot be read in from the configuration file to Excel.

4. Select the "Performance Insight" option under the server node.
5. Click the "Delete" button next to the option.
6. Confirm your input.

Result

The option is removed from the connection settings.

Next, remove all data source items of this option from the report template.

Reloading an option

When a configuration file is loaded, all options available in the file are loaded.

To reload an option that was deleted in the connection settings but is available in the configuration file, select the server node and click "Load".

19.2.4.4 Configuring report templates

Requirement

An online connection or offline connection has been established.

Procedure

To create a new report template, proceed as follows:

1. Open a new Excel file.
2. Add a segment.
   You can choose between time series segments and single value segments.
3. Add data source items to the segment.
   The exact procedure depends on the type of the data source item.
4. Optional: If you do not want a data source item to use the default configuration, determine its configuration.
   You have the following options:
   - Select an existing configuration.
   - Create a new configuration and select it.
   - Define a local configuration.
5. Optional: To define additional segments, repeat steps 2 to 4.
6. Optional: When using an online connection, test the template by reading the runtime data of selected segments or all segments.

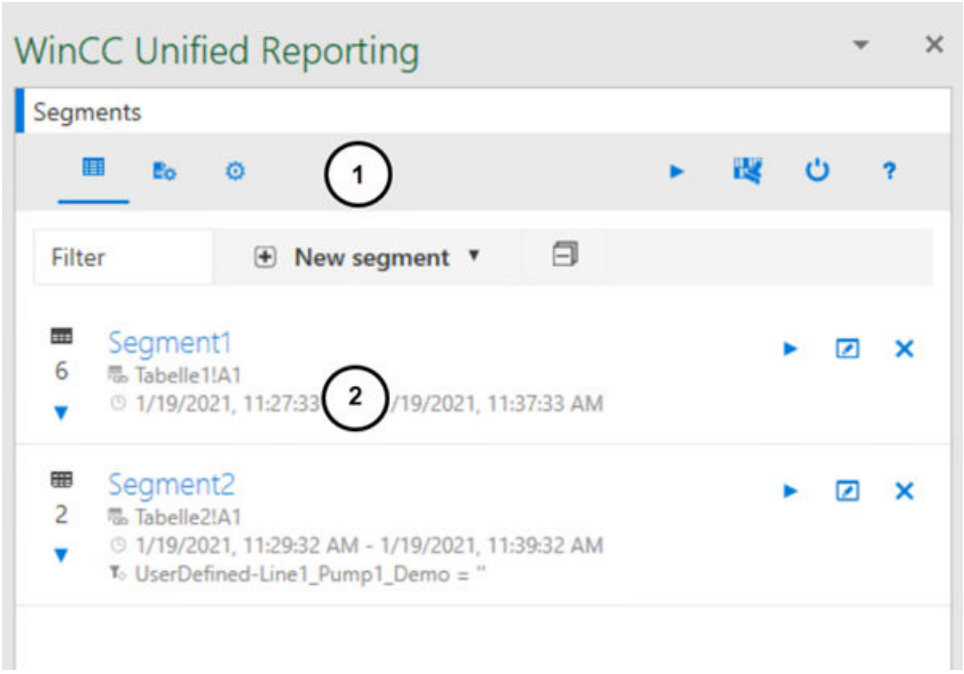
User interface of the add-in

Requirement

- The "WinCC Unified" tab is visible in Excel.

Structure

If you click on "Segments" in the "Configuration" group, you see the following interface:



① Toolbar
② Work area

Toolbar buttons:

Button	Tooltip	Description
	"Segment configuration"	Loads the interface to add and edit segments in the work area.
	"Data source item configuration"	Loads the interface for adding and editing the configuration of a data source item in the work area.
	"Basic settings"	Loads the interface for setting the language settings in the work area.
	"Update all"	Reads the Runtime data of the connected data source into the data tables of the segments.
	"Delete Runtime data"	Removes all Runtime data from the report template.
	Logoff	Logs out the user currently logged in to the add-in.
	Help	Opens the user help for the add-in.

Working with segments

Basic information on segments

Definition

A report template consists of any number of segments. Each *segment* is a container to which you can add any number of data source items. The segment reads the data from its data source items.

There are time series segments and single value segments.

Note

Hierarchical segments of PI options

Hierarchical segments are also available with PI Options installed. For more information on this, refer to the PI Options help.

Time series segments

A *time series segment* documents the data of its data source items in a defined time period.

It has a legend table and a data table.

Data source items

Time series segments can have the following data source items:

- Logging alarms
- Alarm statistics
- Logging tags
- User-defined columns
- Contexts
- Audit

Note

Data source items of the PI options

If PI options are installed, additional data source items can be added. For more information on this, refer to the PI Options help.

Legend table

The table header row provides general information about the segment and its data source items.

You decide which type of information is provided when you create or edit the segment.

Data table

The data table outputs the data of the data source items. It documents how the data source items have changed in the defined time period.

The data table of a time series segment has the following columns:

Columns		Description
Time stamp column		Always output Always output as the first column
Per data source item	Standard column	The standard column provides the standard property of the data source item. This property depends on the type of data source item. For a data source item of the Tag type, e.g. the tag value
	Optional columns	Provide more information about the data source item. What information this is depends on the type of the data source item. For a data source item of the Tag type, e.g. the quality code of the tag value You change the default settings for visibility, column title and order of these columns in the configuration of the data source item.

In the default setting, the data source items in the data table have the order in which they were added to the segment.

Note

When the standard columns and optional columns provide numerical values, you can have the actual values replaced with texts or graphics from a text list or graphic list when importing the Runtime data.

Single value segments

A *single value segment* documents exactly one value for its data source items.

Data source items

Single value segments can have the following data source items:

- Logging tags
- Tags

Note

Data source items of the PI options

If PI options are installed, additional data source items can be added. For more information on this, refer to the PI Options help.

Data table

The data table of a single value segment has the following columns per data source item:

Columns	Description
Standard column	The standard column provides the standard property of the data source item. For tags and logging tags: the tag value
Optional columns	Provide more information about the data source item. For tags and logging tags: • Time stamp • Data source item • Quality code of the tag value You change the default settings for the visibility of these columns in the configuration of the data source item.

The data table of a single value segment shows the data source items in the order in which they were added to the segment.

Note

When the standard columns and optional columns provide numerical values, you can have the actual values replaced with texts or graphics from a text list or graphic list when importing the Runtime data.

Single row segments do not have a table header row. However, in the configurations of their data source items, you can determine whether a caption is inserted for the displayed columns and the position at which this occurs.

See also

Standard column (Page 7672)

Standard column

Introduction

For each data source item of a segment, a standard column is added in the data table of the segment.

Content of the standard column

The standard column provides the standard property of the data source item and depends on the type of the data source item:

Data source item type	Default column title	Value
Logging alarm	"Alarm ID"	Alarm IDs of the displayed alarms
Alarm statistics	"Alarm statistics [ID]"	Alarm IDs of the alarms displayed in the alarm statistics

Data source item type	Default column title	Value
Tag or logging tag	"<Name of the tags or logging tags>"	Value of the tag or logging tag
Context	"<Name of the context object>"	Context name
Audit	"Audit [<object name>]"	The name of the object monitored by the Audit Trail
User-defined column	Name entered when creating the data source item	As set in the configuration of the data source item: • A fixed string. Or • A dynamically calculated string

Changing the column title

You can replace the default column title with a localizable display name. See Setting a display name for standard column (Page 7711).

Replacing numerical values

If the standard column provides numeric values, you have the option to have the actual values replaced with texts or graphics from a text list or graphic list when the Runtime data is read in. See Assigning text lists and graphic lists (Page 7708).

User-defined columns are excluded from this.

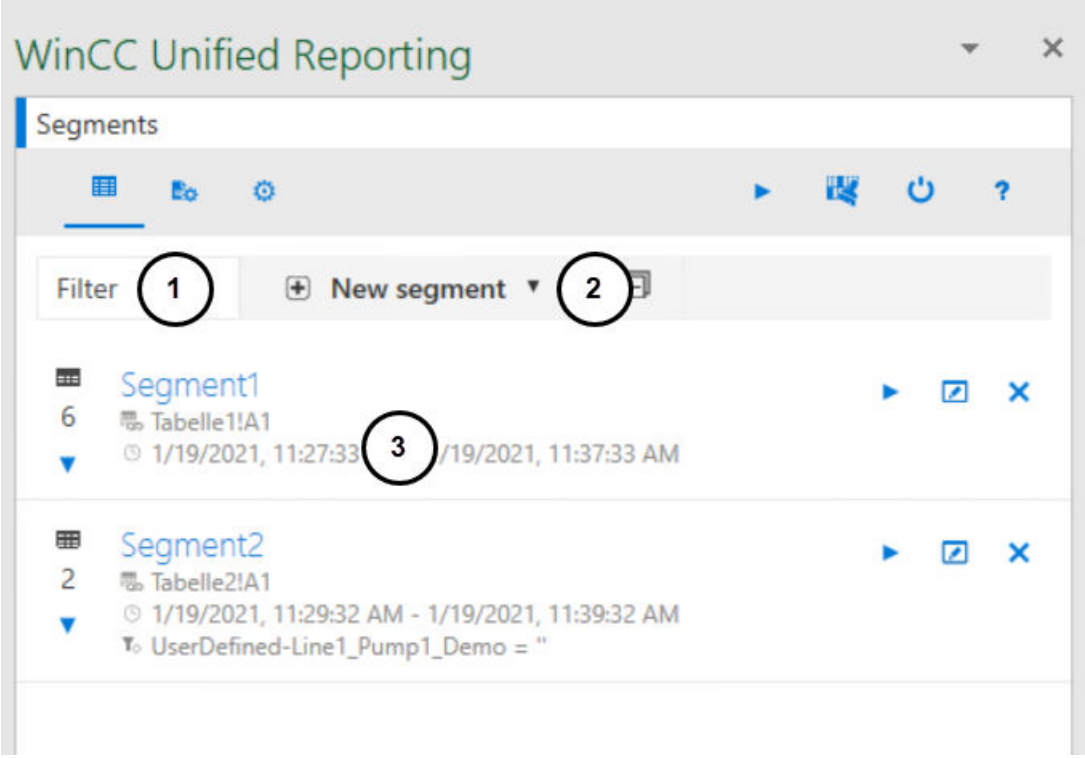
See also

Basic information on segments (Page 7670)

The segment user interface

Structure

The interface for creating and editing segments has the following structure:



① Filter
Filters the list of segments by name.

② Button for creating a segment

③ List of segments
Each segment has buttons for reading in, editing and deleting the segment.
The following configuration is displayed for each segment:

- Segment name
- Number of data source items
- Insertion location of the segment in the Excel file
- Time span
- If context filters have been configured: The filter string

A click on the segment opens the area with the data source items.

Create segments

Requirement

- The "WinCC Unified" tab is visible in Excel.
- The data source is set up.
- To filter the time interval of the time series segment depending on the context: There are contexts in the project that run on the connected Runtime server or are the basis of the configuration file.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "New segment".
3. Select "New time series segment" or "New single value segment".
4. Enter a segment name.

Note

Note the Excel restrictions for naming tables (for example, do not use blanks).

Change the segment name only via the add-in, not via the Excel property "Table name".

Do not change the name of the worksheet after creating the segment. The add-in addresses the segment by the segment name and the worksheet name.

5. For a time series segment, make the following settings in addition:
   - Under "File location" you determine where the segment will be inserted in the file. Enter the name of the worksheet and the cell.
     Alternatively, click "Select a cell" and use the cell currently selected in the Excel file:

     
   - Under "Start" and "End", you determine the time period for which values are read into the segment.

	"Absolute date/ time"	Select a date and a time. The information is absolute to the current date.
	"Relative date/ time"	Select a reference time and a time interval. The information is relative to the current date. See also Formats for relative time information (Page 7679).
	"Date/Time of the cell"	Applies the value of the cell currently highlighted in the Excel file. Make sure that the cell supplies a valid time.
	"Date/Time of the tag"	Applies the value of the set tag. Make sure that the tag supplies a valid time. Possible data types: • DateTime • String • Integer

– (Optional) Under "Properties of the legend table", you configure the contents to be displayed in the table header row of the segment:

"Name" "Start" "End" "State"	General information on the segment
"Context filter"	If the segment time was limited by a context filter: The filter string is output. See step 6.
"Audit status"	If the segment has an audit data source item, the field shows the overall status of the audit data: • Green field: No manipulations of the Audit Trail were found in the queried time range. • Red field: Manipulations of the Audit Trail were found in the queried time range. Single or multiple entries have been deleted, added or changed.
"Header"	The table header row includes a list of the segment's data source items showing general information about these data source items. The information displayed for the data source items depends on their type. Example of contexts: Display name of the context, context provider, hierarchy path, short name of the context, full name of the context, option

Use the check boxes to remove information from or add information to the legend table. To change the sorting in the table header row, move the mouse pointer to a row and shift it using the arrow buttons or drag-and-drop.

6. (Optional) You can filter the time interval of the time series segment depending on the context. You can define up to two filter conditions.
   Proceed as follows:
   - Under "Context filter", click "+" or "Add new condition row".
     The condition line is added.
   - Click on "+" in the condition line.
   - Under "Select context", select the root of the common plant model.
     In the next row, you see the top level of the common plant model.
   - Navigate through the common plant model to plant objects with contexts.
     Plant objects and contexts can be recognized by the following icons:

	Plant object
	Context

   - Select a context.
   - Select an operator.
   - Enter a value.
   - (Optional) Use "+" or "Add new condition row" to create a second condition and select whether the two conditions are to be linked with a logical AND or OR.
7. (Optional) Under "Autofit", configure whether the column width and row height of the data table is automatically adapted to the text read from Runtime.
8. Confirm your entries with "OK".

Result

The segment is created and added to the list of segments:

Next, add data source items to the segment. Your procedure depends on the type of the new data source item.

See also

Tips on design and layout (Page 7717)

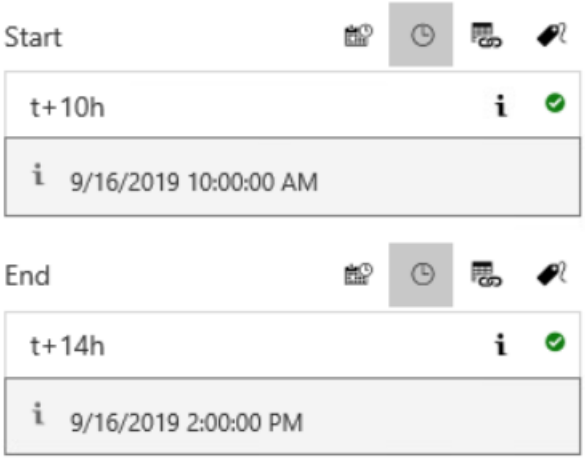
Adding data source elements (Page 7681)

Working with configurations (Page 7694)

Formats for relative time information

Definition of a relative time information

The relative times are entered using a reference time and a time interval.



Reference time

Use one of the following characters for the reference time:

- "*" - Now
- "t" (today) - Today at midnight
- "y" (yesterday) - Yesterday at midnight
- "1-31" - Specific day of the current month

Time interval

- "y" (year): +1y = plus 1 year
- "mo" (month): +1mo = plus 1 month
- "w" (week): +1w = plus 1 week
- "d" (day): +1d = plus 1 day
- "h" (hour): +1h = plus 1 hour
- "m" (minute): +1m = plus 1 minute
- "s" (second): +1s = plus 1 second
- "ms" (milliseconds): +250ms = plus 250 milliseconds

Examples

- *-1y: One year ago today
- t+8h: Today at 8:00 am
- y+8h: Yesterday at 8:00 am
- 1+8h: The first day of the current month at 8:00 am
- *-1d: One day ago
- *-2h-30m-30s: 2 hours, 30 minutes and 30 seconds ago

See also

Create segments (Page 7675)

Edit segments

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Edit" next to a segment in the list of segments.
3. Edit the segment.
   You can make the same settings as when creating the segment.

Delete segments

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Delete" next to a segment in the list of segments.
3. Confirm your entries with "OK".

Adding data source elements

Adding log alarms

Requirement

- There are logging alarms in the project that runs on the connected Runtime server or is the basis of the configuration file.
- The "Alarm" option is enabled in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Adding logging alarms

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded into the add-in.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Alarms" option.
5. Select the "Alarm" entry under "Select alarms".
6. Under "Select alarms", select the entry "Alarm [ID]".

   Note

   Change selection criteria

   After you have added alarms, you can change the selection criteria and add more data source items to the segment.

   For example: Output tags and alarms in the same segment.

7. To cancel your selection, select the entry "Alarm [ID]" under "Selected data source items" and click "Delete".
8. Confirm with "OK".

Result

- The data source item for logging alarms is added to the add-in below the segment.
- The configuration of the data source item controls which data is added when importing the runtime data into the data table.

  Note

  With the default setting, the data source item uses the default configuration. It shows all logging alarms of the project.

To display data that deviates from the default configuration, select one of the following options:

- Select a different matching configuration.
- Create your own configuration.
- Edit a configuration.
- Overwrite a configuration locally.

See also

Creating or editing configurations for log alarms (Page 7694)

Select configuration (Page 7705)

Working with configurations (Page 7694)

Adding alarm statistics

Introduction

To output statistical calculations for logging alarms in a report, add alarm statistics to a report template. The following calculations are available:

- Frequency of an alarm
- Average display time per state machine
- Total display time per state machine
- Maximum display time per state machine
- Minimum display time per state machine

The alarm statistics add columns with statistical calculations and columns with general alarm properties of the recorded alarms to the reports.

You can find more information about calculations in alarm statistics in the help for the alarm control.

Requirement

- The "Alarm" option is enabled in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a time series segment.
   The segment is extended by the area for the data source items.

3. Click "+".
4. Select the "Alarm" option.
5. Under "Select alarms", select the entry "Alarm statistic [ID]".
6. Under "Select alarm statistic" select the entry "Alarm statistic [ID]".

Note

Change selection criteria

After adding the alarm statistics, you can change the selection criteria and add more data source items.

7. (Optional) To cancel your selection, select the entry "Alarm statistic [ID]" under "Selected data source items" and click "Delete".
8. Confirm with "OK".

Result

The added data source item for alarm statistics is displayed below the segment and inserted into the data table.

First, the data table shows the contents configured in the default configuration for alarm statistics. To output other contents, select or create a configuration.

See also

Creating or editing configurations for an alarm statistics (Page 7696)

Working with configurations (Page 7694)

Add logging tags

Requirement

- The project on which the connected Runtime server runs or the basis of the configuration file has logging tags.
- The "Logging tag" option was selected while setting up of the connection.
- The "WinCC Unified" tab is visible in Excel.
- A single value segment or time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".

4. Select the "Logging tag" option.
5. Optional: To reduce the load time, filter which tags are loaded to the selection under "Add filter".
   The preset filters "*" return all logging tags of the project.
   – "Tag name": Enter the name of the tag whose logging tags you want to add.
   – "Logging tag name": Enter the name of the logging tags you want to add.

   Note that the entry is case-sensitive.

   Note

   Filter by partial string

   You use the wildcard "*" to filter by partial strings.

   For example:
   - *T* returns all tags with a "T" in their name.
   - *T returns all tags that end in "T".
   - T* returns all tags that start with "T".

   When filtering for structures, the separators must be part of the filter string.

   For example: The following filters return the logging tags for all tags of the device "HMI_RT_1":
   - Filter for tag: "HMI_RT_1::*"
   - Filter for logging tag: "*"
6. Click "Load".
   The logging tags of the project are filtered and provided under "Select tags".
7. Optional: Further reduce the number of tags that are offered for selection by clicking next to "Select logging tags" and entering another filter string.
   The list of tags you are being offered is filtered while you type.
8. Select one or more tags under "Select logging tags".
   The tags are added to the "Selected data source items" list.

   Note

   Change selection criteria

   After you have added a tag, you can select a different option or a different filter and add additional data source items.

   For example: Output KPIs and logging tags in the same segment.
9. To remove one or more data source items from "Selected data source items", select them and click "Delete".
10. Confirm with "OK".
    The added logging tags are shown below the segment and added to the Excel table.
11. If you have added the logging tag to a single value segment:
    – In the Excel worksheet, select the cell in which the logging tag is to be inserted.
    – Click the "Select a cell" button on the data source item of the logging tag.
      Alternatively, enter the name of the worksheet and the cell.

See also

Create segments (Page 7675)

Select configuration (Page 7705)

Create or edit configurations for logging tags (Page 7697)

Working with configurations (Page 7694)

Adding tags

Requirement

- The project on which the connected Runtime server runs or the basis of the configuration file has tags.
- The "Tag" option was enabled when the connection was set up.
- The "WinCC Unified" tab is visible in Excel.
- A single value segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select the single value segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Tag" option.
5. Optional: To reduce the load time, filter which tags are loaded to the selection under "Add filter".
   Under "Tag name", enter a filter, e.g. the name of the tag. Note that the entry is case-sensitive. The default filter "*" returns all tags of the project.

   Note

   Filter by partial string

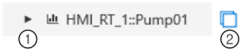
   You use the wildcard "*" to filter by partial strings.

   For example:
   - *T* returns all tags with a "T" in their name.
   - *T returns all tags that end in "T".
   - T* returns all tags that start with "T".

   When filtering for structures, the separators must be part of the filter string.

   For example: The filter "HMI_RT_1::*" returns all tags of the device "HMI_RT_1".

6. Click "Load".
   The tags of the project are filtered and provided under "Select tags".
   You can recognize structs and arrays in the list by the following items:



① Button to display the members of the struct or array
② "Select all included data source items"
Button that adds all members with a simple data type to the list of selected data source items

7. Optional: Further reduce the number of tags that are offered for selection by clicking next to "Select tags" and entering another filter string.

   The list of tags you are being offered is filtered while you type.

8. Select which tags will be added to the segment. You have the following options:

Target	Procedure	Result
Show the members of a struct or array.	Click the button with the arrow next to the struct or array.	A second "Select tags" list is added, in which you can see all the members of the struct or array. You can add to the segment any members that have a simple data type, e.g. bool, float or string.
Add all members of a struct or array.	Next to the struct or array, click "Select all included data source items".	All members with a simple data type are added to the "Selected data source items" list and marked as selected under "Select tags":
Select tags with simple data type.	Under "Select tag", click the required tags.	The tags are added to the "Selected data source items" list and marked as selected under "Select tags": 

Note

Automatic filtering when displaying the members or selection of all members

If you click the button to display the members of a struct or array or activate the option to select their members, the struct or array is set as a filter:

- The list under "Select tags" only shows the struct or array.
- A second "Select tags" list is added below this, in which you can see all members of the struct or array.

To see all available tags again, delete the filters.

Note

Change selection criteria

After you have added a tag, you can select a different option or a different filter and add additional data source items.

9. To remove tags from the segment, click on the tags in "Selected data source items" and click "Delete".
10. Confirm with "OK".

The added tags are added to the segment.

When the report template is updated in the add-in and when the report is generated in runtime, the tag values are inserted into the data table.

See also

Creating or editing configurations for tags: (Page 7699)

Working with configurations (Page 7694)

Adding contexts

Introduction

To display in a report which contexts are to run during a certain time period, add only contexts to a segment in the report template.

To display which process data has been accumulated during the runtimes of a context, add the context and other data source items, such as logging tags or logging alarms, to the segment.

Requirement

- There are contexts in the project that run on the connected Runtime server or are the basis of the configuration file.
- The "Context" option is enabled in the connection settings.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Adding a context to a segment

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Context" option.

5. Select a context:
   - Under "Select a context definition", select the root of the plant model.
     In the next row, you see the top level of the common plant model.
   - Navigate through the common plant model to plant objects with contexts.
     Plant objects and contexts can be recognized by the following icons:

	Plant object
	Context

   - Select the desired contexts.
     All selected contexts are included in the "Selected data source items" list

Note

Change selection criteria

After you have added a context, you can select a different option and add additional data source items.

For example: Context and logging tags in the same segment.

6. To remove one or more data source items from "Selected data source items", select them and click "Delete".
7. Confirm with "OK".

Result

The selected contexts are displayed below the segment and inserted into the data table.

If you do not want a context to use the default configuration, select its configuration next.

Content of the data table after executing the segment

In segments to which only contexts or contexts and user-defined columns have been added:

- A line is inserted for each context whose runtime falls within the time period of the segment.
- "Time stamp" column: The time at which the context was started

In segments that combine contexts with logging tags or logging alarms:

- All logged values with the same time stamp are displayed per row.
- "Time stamp" column: The logging event
- "Start time" column: The time at which the context was started
- "Context " <Context name>"" column: The value passed to the context at start
- If no context was started at the time of logging, the context cells remain empty.

Example

The following data source items were added to a segment:

- The "Product" context
  Runtime of the context: 15:00:00 to 19:59:59 hours
  The context was started with the "Orange lemonade" value.
- The "Logged_Rotation" logging tag
  Logging cycle: 2s
- The "Logged_Temperature" logging tag
  Logging cycle: 5s
- The user-defined "Unit" column
  It contains the unit for "Logged temperatures".

Content of the data table after execution of the segment:

	A	B	C	D	E	F	G
1	Time stamp	Context "Product"	.../Line1-Product:StartTime	.../Line1-Product:EndTime	Logged_Rotation	Logged_Temperature	Unit
2	Mo, 24 Feb 2020 15:00:00,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	10	20	°C
3	Mo, 24 Feb 2020 15:00:02,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	100		°C
4	Mo, 24 Feb 2020 15:00:04,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	500		
5	Mo, 24 Feb 2020 15:00:05,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99		40	°C
6	Mo, 24 Feb 2020 15:00:06,100	Orange lemonade	Mo, 24 Feb 2020 15:00:00,00	Mo, 24.02.2020 19:59:59,99	750		°C
7	...	...	...	...	...	...	...
8	Mo, 24 Feb 2020 20:00:00,100				650		°C

Lines 2 to 6	Values were logged for "Logged_Rotation" and "Logged_Temperature", while the "Product" context ran with the "Orange lemonade" value.
Line 8	A value was logged for "Logged_Rotation" while no context was running.

Adding user-defined columns

Introduction

User-defined columns supplement the data of the other data source items of a time series segment with additional information:

- With a fixed string
  The string appears in each cell of the column.
  Example: Display measurement unit of the tag values in report
- With a formula
  The formula is calculated during generation for each cell in the dynamic column.
  Example: The sum of the tag values output in the report.

The configuration of the user-defined column controls which string or formula it uses.

Requirement

- The "User-defined column" option was enabled when the connection was set up.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list of segments is loaded.
2. Select a segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the option "User-defined column".
5. Enter the name of the column under "name".
6. Click "Select" or press <ENTER>.
   The column is included in the list "Selected data source items".

   Note

   Change selection criteria

   After you have added a column, you can select a different option or a different filter and add additional data source items.

7. Select a configuration for the user-defined column.
8. To remove one or more data source items from "Selected data source items", select them and click "Delete".
9. Confirm with "OK".

The added columns are displayed below the segment and inserted into the data table.

See also

Creating and editing configurations for user-defined columns (Page 7701)

Working with configurations (Page 7694)

Add Audit

Introduction

To output the Runtime device Audit Trail in a report, add an Audit data source item to a report template.

You can find more information about the Audit option in WinCC Unified in the TIA Portal help.

Requirement

- The Audit option was activated in the engineering for the Runtime device.
- The "Audit" option is activated in the connection settings of the Excel add-in.
- The "WinCC Unified" tab is visible in Excel.
- A time series segment is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
   The list with the segments already created is loaded.
2. Select a time series segment.
   The segment is extended by the area for the data source items.
3. Click "+".
4. Select the "Audit" option.
5. Select the Audit Trail.
6. (Optional) To undo your selection, select the Audit Trail under "Selected data source items" and click "Delete".
7. Confirm with "OK".

Result

The Audit data source item is displayed below the segment.

If an Audit Trail is configured for the data source, the Audit data is inserted into the report when the Runtime data is read into Excel and when it is generated in Runtime:

- In the legend table: Identifier of the overall status of the Audit Trail for the queried time range in the "Audit Status" field

Value	Description
Green	No manipulations of the Audit Trail were found in the queried time range.
Red	Manipulations of the Audit Trail were found in the queried time range. Single or multiple entries have been deleted, added or changed.

Requirement: The "Audit status" option is activated on the segment under "Header properties".

Note

Overall status for check mode "None"

If the check mode "None" is set in the configuration of the audit data sources item, the "Audit status" field is always green.

- In the data table of the segment: Identifier of manipulations

Type of manipulation	Identifier in the data table
Value of entries changed	Directly at the entries
Entries added	
Entries deleted	The manipulated time range receives a start and end entry.

First, the data table shows the contents configured in the standard configuration for Audit. To output other contents, select or create a configuration.

Setting a display name for standard column

Introduction

You can assign a display name for the standard column of a data source item. When a display name is set, it is used in the data table instead of the default column title and is listed in the table header row.

Display names make reports clearer, for example, when data source items for contexts or tags have very long names.

You set the display name in the local configuration of the data source item.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- There is a segment with a matching data source item.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Expand a segment by clicking on it.
   The area for adding and editing data source elements appears.
3. Move the mouse pointer to the data source item and click "Edit".
   The local configuration of the data source item opens.
4. Enter the desired column title in "Display name".
   The column title must be unique within the segment.

   Note

   Localization

   The column title is stored in the Runtime language currently set in the basic settings of the add-in.

   To localize the column title, change the Runtime language and repeat your entry in the new language.

5. Confirm your entry with "OK".

Result

- The data table uses the display name as the column title for the standard column of the data source item.
- As long as the following conditions are met, the "Display name" column is inserted into the table header row:
  - Display of the header row in table header row is enabled.
    Make this setting at the segment.
  - A display name is set for at least one data source item of the segment.

  The column lists the display names of all data source items. If no display name is set for a data source item, its cell remains empty.

Note

- If you assign a different configuration to the data source item, the display name is retained.
- To return to the display of the default column title after assigning a display name, enter the name of the data source item under "Display name".

Delete data source elements

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source element is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Expand a segment by clicking on it.
   The area for adding and editing data source elements appears.
3. Move the mouse pointer over a data source element and click "Delete".

Working with configurations

Basics of configuration

The *configuration* of a data source item defines the values of a data source element that are displayed in a segment or how they are calculated and displayed.

There are specific configuration settings for each data-source-item type.

Data source items used in time series segments use a different configuration than data source items used in single-value segments.

You have the following options:

- Use standard configuration.
  There is a standard configuration for all types of data source items. Once added, data source items use the default configuration of their type.
  You can edit the standard configurations.
- Use user-defined configuration.
  You can create any number of user-defined configurations for all types of data source items.
  You can select one of the user-defined configurations on the data source item.
- Overwrite a configuration locally.
  You can overwrite the configuration selected at the data source item locally.

Creating or editing configurations for log alarms

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > Logging alarm configuration".
4. Enter the name of the configuration under "Configuration name".

5. (Optional) Enter texts or graphics from a text list or graphic list in the standard column instead of the alarm IDs.
   See Assigning text lists and graphic lists (Page 7708).
6. (Optional) Change the default settings of the optional columns. The optional columns are used to display the alarm properties.
   See Configuring optional columns (Page 7707).
7. (Optional) Filter the logging alarms to be displayed. You define a filter query for this purpose. The filter query can consist of up to two conditions.
   Proceed as follows:
   - Under "Filter", click "+" or "Add new condition row".
   - Select an alarm property, an operator, and enter a value.
   - Optional: Use "+" or "Add new condition row" to create additional conditions. Select whether the conditions are to be linked with a logical AND or OR.
8. Enable the option "Use system colors" so that the alarms are highlighted with the same colors as in the alarm control.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7711).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for logging alarms.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

See also

Select configuration (Page 7705)

Calculation modes for data source elements (Page 7715)

Creating or editing configurations for an alarm statistics

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > Alarm statistics configuration".
4. Enter the name of the configuration under "Configuration name".
5. (Optional) Enter texts or graphics from a text list or graphic list in the standard column instead of the alarm IDs.
   See Assigning text lists and graphic lists (Page 7708).
6. (Optional) Change the default settings of the optional columns. The optional columns are used to display the statistical calculations and alarm properties.
   See Configuring optional columns (Page 7707).
7. (Optional) Filter the contents to displayed in the alarm statistics. You define a filter query for this purpose. The filter query can consist of up to two conditions.
   Proceed as follows:
   - Under "Filter", click "+" or "Add new condition row".
   - Select an alarm property, an operator, and enter a value.
   - Optional: Use "+" or "Add new condition row" to create additional conditions. Select whether the conditions are to be linked with a logical AND or OR.
8. Enable the option "Use system colors" so that the alarms are highlighted with the same colors as in the alarm control.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item.

See Setting a display name for standard column (Page 7711).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for alarm statistics.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

See also

Adding alarm statistics (Page 7682)

Create or edit configurations for logging tags

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration".
4. To create a configuration for logging tags in a time series segment, select the entry "Logging tag configuration".
   To create a configuration for logging tags in a single value segment, select the entry "Single value configuration logging tag".
5. Enter the name of the configuration under "Configuration name".
6. Under "Calculation mode", select the data to be written if no current value is available.
7. (Optional) If the configuration is for logging tags with the numeric data type, you can output texts or graphics from a text list or graphic list in the standard column instead of the tag value. See Assigning text lists and graphic lists (Page 7708).
8. Set the other settings as described below.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7711).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for logging tags.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Additional settings for time series segments

In time series segments, the following additional settings are available for logging tags:

Setting	Description
"Interval"	Only for the "Stepped" and "Interpolated" calculation modes.
"Columns" > "Quality Code"	(Optional) Change the default settings of the optional "Quality Code" column. See Configuring optional columns (Page 7707).

Additional settings for single value segments

In single value segments, the following additional settings are available for logging tags:

Setting	Description
"Time stamp"	Determine the date and time for which the value is read. Proceed as described below.
"Show captions"	Define whether a header is displayed in the columns for the time stamp, the data source item and the quality code.
"Show time stamp"	Determine whether and where this information is displayed in the table. The information is always in relation to the value cell.
"Show data source item"	
"Show quality code"	

To set the "Time stamp", select one of the following options:

	Absolute time information	Select a date and a time. The information is absolute.
	Relative time information	Select a reference time and a time interval. The information is relative to the current date.

	Read time information from cell	Applies the value of the cell currently highlighted in the Excel file. Make sure that the cell supplies a valid time.
	Read time information from tag	Applies the value of the set tag. Make sure that the tag supplies a valid time. Possible data types: • DateTime • String • Integer

See also

Calculation modes for data source elements (Page 7715)

Creating or editing configurations for tags:

Requirement

- The "WinCC Unified" tab is visible in Excel.

Creating a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":
3. Click "New Configuration> Tag single value configuration".
4. Enter the name of the configuration under "Name".
5. (Optional) If the configuration is for tags with the numeric data type, you can output texts or graphics from a text list or graphic list in the standard column instead of the tag value. See Assigning text lists and graphic lists (Page 7708).
6. Set the other settings as described below.
7. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7711).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration".

3. Click a configuration for tags.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Settings for single value segments

In single value segments, the following settings are available for tags:

Setting	Description
"Show captions"	Select whether a header is displayed in the columns for the time stamp, the data source item and the quality code.
"Show time stamp"	Select whether the time stamp is output with the value.
"Show data source item"	Select whether the quality code is output with the value.
"Show quality code"	Select whether the quality code is output with the value.

Creating or editing configurations for contexts

Requirement

- The "WinCC Unified" tab is visible in Excel.

Core statement

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration".
4. Select the entry "Configuration context".
5. Enter the name of the configuration under "Configuration name".
6. (Optional) Change the default settings of the optional columns. The optional columns are used to display important contextual information.
   See Configuring optional columns (Page 7707).
7. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7711).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click a configuration for contexts.
4. Edit the configuration settings.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Creating and editing configurations for user-defined columns

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > User-defined column configuration".
4. Enter the name of the configuration under "Configuration name".

5. Under "Formula", select one of the following options:
   - Enter a fixed string.
     The string is transferred into each cell of the column.
   - Enter an Excel formula.
     The formula is copied into each cell of the user-defined column and adapted to the respective row.
     To prevent a part of the formula from being adjusted, place the character "$" in front of it.
     Example

Formula in configuration		=B2+C2	=B$2+C2
Adapting the formula in the report	in line 2	=B2+C2	=B2+C2
	in line 3	=B3+C3	=B2+C3
	in line 4	=B4+C4	=B2+C4

Note

No validity check

The formula is not tested for correctness during either input or dynamic adaptation.

6. Confirm your entries with "OK".

Adding or editing configurations for audit

Introduction

Check mode

The check mode of the configuration of an audit data source item determines

- Whether an integrity check is performed when the Runtime data is read, and what is checked. You can output the overall result of the check in the table header row in the "Audit status" field.
- Which audit data records are provided in the data table.

Possible check modes:

"None"	Provides the data for all audit data records that fall within the requested time range. No integrity check is performed. Default setting
"Check only"	Checks all audit data records that fall within the requested time range without providing their data. It is tested whether data records have been manipulated, deleted or added.
"Check entries"	Check the audit data records and provides their data that fall within the queried time range and that have not been deleted from the Audit Trail or subsequently added. It is checked whether data records have been manipulated.
"Check all"	Checks all audit data records and provides their data that fall within the queried time range. It is tested whether data records were manipulated, deleted from the audit trail or subsequently added.

Filter type

An Audit data record consists of two entries:

- An entry for the user expectation
- An entry for the system response.

User expectation and system response may differ. In addition, there are situations in which only one of the two data records is created.

The filter type controls which data records and which entries are included in the report.

Possible filter types:

Filter type	User expectation equals system response	User expectation does not equal system response	Data record entry for user expectation or system response is missing
"Show all data in detail"	Both data record entries are inserted.		The existing data record entry is inserted.
"Show data and conformity errors"	The data record entry with the user expectation is inserted.	Both data record entries are inserted.	
"Show only data with conformity errors"	No data record entry inserted.		

Requirement

- The "WinCC Unified" tab is visible in Excel.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

3. Click "New Configuration > Audit configuration".
4. Enter the name of the configuration under "Name".
5. Select a check mode:
6. Specify a filter type.
   Preset value: "Show data and conformity errors"
7. (Optional) Change the default settings of the optional columns. The optional columns are used to display the audit attributes.
   See Configuring optional columns (Page 7707).

8. (Optional) To further filter the inserted content, define a filter query.
   The filter query can consist of up to two conditions. Proceed as follows:
   - Under "Filter", click "+" or "Add new condition row".
   - Select an Audit attribute, an operator and enter the value of the attribute.
   - Optional: Use "+" or "Add new condition row" to create additional conditions. Select whether the conditions are to be linked with a logical AND or OR.
9. Confirm your entries with "OK".

Note

To not use the default column title for the standard column, set a display name in the local configuration of the data source item. See Setting a display name for standard column (Page 7711).

Editing a configuration

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration":

   
3. Click a configuration for Audit.
4. Edit the configuration settings. You have the same options as when creating the configuration.
5. Confirm your entries with "OK".

The changes are applied the next time you read in the Runtime data.

Examples of the configuration of the filter type

The following table contains examples of data records that were generated in Runtime through changes to tags monitored by Audit:

Data record ID	Tag name	Modified by	Old value	New value	Description
1A	Motor1_Speed	User1	0	10	An operator changes the speed of a motor in an I/O field of an HMI screen.
1B	Motor1_Speed	System	0	10	User expectation and system response are identical.
2A	ValvePercentile	User1	0	100	An operator opens a valve using a slider on an HMI screen. The valve has a physical blockage and cannot be opened. Therefore, no data record entry for the system response is generated.

Data record ID	Tag name	Modified by	Old value	New value	Description
3A	ValvePercen-tile	User1	0	99	A physical block has been removed and the operator repeats the entry. The valve reacts, but cannot be fully opened. User expectation and system response differ.
3B	ValvePercen-tile	System	0	49	
4B	Mo-tor2_Speed	System	0	20	An operator changed the speed of another motor. The resulting data record was manipulated, and the user expectation entry was deleted. There is only one entry for the system response.

The following table shows which data record entries are inserted into the data table depending on the filter type selected when generating the report:

Data record ID	Tag name	Modified by	Old value	New value
Filter type "Show all data in detail"				
1A	Motor1_Speed	User1	0	10
1B	Motor1_Speed	System	0	10
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20
Filter type "Show data and conformity errors"				
1A	Motor1_Speed	User1	0	10
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20
Filter type "Show only data with conformity errors"				
2A	ValvePercentile	User1	0	100
3A	ValvePercentile	User1	0	99
3B	ValvePercentile	System	0	49
4B	Motor2_Speed	System	0	20

Select configuration

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source item is available.
- There is a user-defined configuration for the type of the data source item.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The segment is extended by the area for the data source items.
3. Select the desired configuration from a data source item in the drop-down list.
4. Click "OK".

Result

The changes are applied the next time you read in the runtime data.

Overwrite a configuration locally

A local configuration overwrites the configuration selected at the data source item. It applies only to the data source item for which it was entered.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A segment with a data source item is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The segment is expanded to include the plant complex for the data source items.
3. Move the mouse over a data source item and click "Edit".
   You create a local configuration that first adopts the values of the original configuration.
4. Enter a name for the local configuration.
5. (Optional) Set a display name. See Setting a display name for standard column (Page 7711).
6. Make the remaining settings as required.
   You can make the same settings as in the default and custom configurations.
7. Confirm your entries with "OK".

Result

The changes are applied the next time you read in the Runtime data.

Delete configuration

Requirement

A configuration is available.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click "Data source item configuration".
3. Move the mouse to a configuration.

   Note
   Default configurations cannot be deleted

   You can edit default configurations but not delete them.

4. Click "Delete".

Result

- The configuration is deleted.
- Data source items with this configuration obtain a local configuration with the same settings.

Configuring optional columns

Introduction

In time series segments, data source items of the following types have optional columns:

- Logging tag
- Logging alarm
- Alarm statistics
- Audit
- Context

The optional columns of a data source item depend on its type. The configuration of the data source items controls whether and how the data table shows these columns.

This section describes how to configure the optional columns.

Requirements

The data source item configuration is open. The configuration must apply to a time series segment.

Showing and hiding columns

1. To show an optional column in the data table, enable the option for the desired column in the "Columns" area.
2. To hide a column, disable its option.

Changing the column title

The data table uses as default column titles the identifiers you see in the "Columns" area. To change the default column titles, do the following.

1. In the "Columns" area, move the mouse pointer to an optional column.
2. Click the button with the pin.
3. Assign a unique column title.

Note

Localization

The column title is stored in the Runtime language currently set in the basic settings of the add-in.

To localize the column title, change the Runtime language and repeat your entry in the new language.

Changing the column sequence

To change the order of the optional columns in the data table, proceed as described in Changing the column sequence (Page 7712).

Assigning text list or graphic list

The values of numeric columns can be replaced by texts or graphics when the Runtime data is read in.

To assign a suitable text list or graphic list to the property, proceed as described in Assigning text lists and graphic lists (Page 7708).

Assigning text lists and graphic lists

Introduction

If standard columns and optional columns of data source items output numerical values, you can assign text lists and graphic lists to these columns. When the Runtime data is read in, the cell values of these columns are replaced by texts or graphics from the assigned lists.

This function improves the readability of the report and helps to draw the reader's attention to important information.

Note

Restrictions

- Tags/logging tags
  Assign a text list or graphic list to the standard column of data source items with a Tag or Logging tag type only if the tag or logging tag has a numeric data type.
  You can assign a text list or graphic list to the optional "Quality Code" column regardless of the data type of the tag.
- User-defined columns
  It is not possible to assign a text list or graphic list for data source items with the User-defined column type.
- Context and audit
  Usually, the names of context objects and audit objects displayed in the standard column do not contain purely numerical values. It is not recommended to assign a text list or graphic list.

Example

Add two data source items with the same logging tag to a time series segment.

For the first data source item, use the default configuration. This causes the report to output the tag value in the standard column.

For the second data source item, select a configuration in which a graphic list is assigned to the standard column. The graphic list contains representational graphics staggered by value range. As a result, the report outputs a graphic in the standard column.

After reading in the Runtime data, the standard column of the second data source item makes readers of the report aware of limit violations. Readers can get the exact tag value from the standard column of the first data source item.

Requirements

- A segment with a data source item was created in the add-in.
- Suitable text lists or graphic lists have been configured in engineering for the connected data source.

Assigning text lists and graphic lists to the standard column

1. Click on "Segments" in the "Configuration" group.
2. Select the segment.
   The data source items of the segment are displayed.
3. Move the mouse over a data source item and click "Edit".
   The local configuration of the data source item opens.

4. Select a suitable list under "Assign text/graphic list".
5. To preview the selected list and its graphics or texts, click the "i" button.
   To hide the preview, click the "i" button again.

Assigning optional columns to text lists and graphic lists

1. Click on "Segments" in the "Configuration" group.
2. Select one of the following options:
   To make the assignment apply to a specific data source item, follow these steps:
   - Select the segment.
     The data source items of the segment are displayed.
   - Move the mouse over the data source item and click "Edit".
     The local configuration of the data source item opens.

   To make the assignment apply to multiple data source items of the same type, follow these steps:
   - Click "Data source item configuration": 
     You can see all default and custom configurations.
   - Click on the desired configuration.
     The configuration opens.
3. In the "Columns" area, click the following button next to the desired optional column:

   An interface for assigning a text list or graphic list is loaded into the add-in.
4. Select the desired graphic list or text list from the drop-down list.
5. To preview the selected list and its graphics or texts, click the "i" button.
   To hide the preview, click the "i" button again.

   Note

   If you are connected offline to the data source, no preview of graphic lists is available.

6. Confirm your entries.

Result

When the Runtime data is read in, the assigned lists are searched for an entry that matches the actual cell value:

- If a matching entry is found, the corresponding text or graphic is inserted into the data table.
- If no matching entry is found, the actual cell value is inserted.

Note

The assignment of graphic lists slows down the import of the Runtime data into the Excel add-in.

See also

Standard column (Page 7672)

Basic information on segments (Page 7670)

Setting a display name for standard column

Introduction

You can assign a display name for the standard column of a data source item. When a display name is set, it is used in the data table instead of the default column title and is listed in the table header row.

Display names make reports clearer, for example, when data source items for contexts or tags have very long names.

You set the display name in the local configuration of the data source item.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- There is a segment with a matching data source item.

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Expand a segment by clicking on it.
   The area for adding and editing data source elements appears.
3. Move the mouse pointer to the data source item and click "Edit".
   The local configuration of the data source item opens.
4. Enter the desired column title in "Display name".
   The column title must be unique within the segment.

   Note

   Localization

   The column title is stored in the Runtime language currently set in the basic settings of the add-in.

   To localize the column title, change the Runtime language and repeat your entry in the new language.

5. Confirm your entry with "OK".

Result

- The data table uses the display name as the column title for the standard column of the data source item.
- As long as the following conditions are met, the "Display name" column is inserted into the table header row:
  - Display of the header row in table header row is enabled.
    Make this setting at the segment.
  - A display name is set for at least one data source item of the segment.

  The column lists the display names of all data source items. If no display name is set for a data source item, its cell remains empty.

Note

- If you assign a different configuration to the data source item, the display name is retained.
- To return to the display of the default column title after assigning a display name, enter the name of the data source item under "Display name".

See also

Standard column (Page 7672)

Overwrite a configuration locally (Page 7706)

Changing the column sequence

Introduction

For a time series segment, you can change the default column order of the data table.

You have the following options:

- Specify the order which the data source items have in the data table.
- For each data source item: Set the order of its optional columns.

Note

The time stamp column always appears first.

Requirement

- The "WinCC Unified" tab is visible in Excel.
- A time series segment has been created.

Change the order of data source items

Procedure

1. Click on "Segments" in the "Configuration" group.
2. Click the time series segment in the list of time series segments.
   The data source items of the segment are displayed.
3. Left-click a data source item and move it up or down using drag-and-drop operation.

Result

The order of data source items in the segment interface is changed.

The next time the Runtime data is read in, the data table outputs the data source items in this order.

Changing the order of optional columns

Procedure

1. Select one of the following options:
   To change the column order of a particular data source item, follow these steps:
   - Select the segment.
     The data source items of the segment are displayed.
   - Move the mouse over the data source item and click "Edit".
     The local configuration of the data source item opens.

   To change the column order for all data source items that use the same configuration, follow these steps:
   - Click "Data source item configuration":

     You can see all default and custom configurations.
   - Click on the desired configuration.
     The configuration opens.
2. Move the mouse pointer to a column under "Columns".
   The columns you see depend on the type of data source item.
3. Move the column up or down using the arrow buttons or drag-and-drop.

Result

The order of the optional columns in the configuration is changed.

The next time the Runtime data is read in, the data table outputs the optional columns in the changed order.

See also

Creating or editing configurations for log alarms (Page 7694)

Basic information on segments (Page 7670)

Configuring optional columns (Page 7707)

Reading Runtime data in Excel

Note

Reading in Runtime data in Excel is used for testing. It is not intended for mass retrieval of data, as is the case when report jobs are executed in Runtime.

Requirement

An online connection is established.

Reading in all segments

1. Select "WinCC Unified > Segments".
2. Click "Update all":

Reading in individual segments

1. Select "WinCC Unified > Segments".
2. Next to a segment in the list of segments click, "Update":

Result

The segment or segments are run. The Runtime data of your data source items are read into Excel.

Note

Controlling the column width and row height

When the automatic adjustment of the column width and row height is disabled in the segment properties, the text read in may be truncated or the formula results are replaced with "#" characters.

Check the column widths and row heights and adjust them manually, if required, or select automatic adjustment. Manual adaptations only apply in the Excel add-in. They are not included in the generated reports.

Note

Removing Runtime data from report template

Remove the Runtime data from the report template before you save the report template and make it available for uploading to Runtime.

To do this, click the "Delete Runtime data" button  in the toolbar of the Excel add-in.

Diagnostics during the data query

Successful execution of the data query is documented by the add-in with a status message in the table.

If an error occurs during the data query, a general error message is displayed under status. In addition, detailed error messages are displayed in the "ErrorLog" worksheet.

Calculation modes for data source elements

If there is no current value for a data source item for a requested point in time, the following calculation modes are available.

Calculation modes for tags

The following calculation modes are available for tags of a time series segment:

Calculation mode	Description
Raw	The actual value available for the specified period. If no data are available, no value is output.
Stepped	If no data are available, the last value is used. With this mode you can also use values with an invalid quality code.
Interpolate	The values are interpolated linearly for the specified time period. With this mode you can only use values with a valid quality code.

The following calculation modes are available for tags of a single value segment:

Calculation mode	Description
Interpolate	The values are interpolated linearly for the specified time period. With this mode you can only use values with a valid quality code.
Left	If no data is available, the last value before the specified period is used.
Right	If no data is available, the last value after the specified period is used.

See also

Create or edit configurations for logging tags (Page 7697)

19.2.4.5 Making general settings

Changing the language

Changing the add-in language

The Excel add-in automatically uses the same user interface language as Excel. If you are using a language for Excel that is not included in the Unified options, English is used as the default language.

You can select the language for the contents of the report independently of the interface. To select another language, the language must be configured in Runtime.

Selecting the language for the report

1. Select "WinCC Unified > Segments".
2. Click "Basic settings":

3. Under "Runtime language", select the language of the report content.
4. Under "Query language" you select which language data queries have that require user input, e.g. filter definitions.

Adapting the work area

Undocking and moving the add-in

To enlarge your working area, you can undock the Excel add-in:

1. Open the drop-down list in the header of the add-in.
2. Click "Move".
3. Move the mouse pointer to the desired location and click the left mouse button.

4. To move the add-in again, keep the left mouse button pressed in the header of the add-in and move the mouse.
5. To dock the add-in again, double-click in the header of the add-in.

Adapting the size of the add-in

1. Open the drop-down list in the header of the add-in.
2. Click "Resize".
3. Move the mouse pointer to the left to make the add-in wider or to the right to make it narrower.
4. Left-click when you have reached the desired size.

Zooming in the add-in

Procedure

To zoom in or out of the display in the add-in, press <CTRL> and move the mouse wheel.

19.2.4.6 Undo and redo

The Excel functions "Undo" and "Redo" are not available in the add-in.

19.2.4.7 Tips on design and layout

This section includes tips on the visual design of reports. The apply for:

- Report templates
- Reports that were generated as XLSX file

Note

Deviating PDF results

A PDF report created by LibreOffice can deviate in content or layout from a PDF report generated with Excel, for example, if the report template uses common Excel features that LibreOffice does not support, such as special fonts or chart types.

Arranging segments

Always place the segments of a report template side by side or each in their own worksheet.

Because the data tables of the segments grow dynamically, tables can overlap when segments are placed one below the other. In the add-in, this causes an error of the OfficeExtension.Error class when reading in the Runtime data.

Changing the column sequence

See section Changing the column sequence (Page 7712).

Adapt column width and row height

For each segment of a report template, check whether the column widths and row heights of your data table are wide or high enough for the values to be read. If this is not the case, texts in the generated report are truncated or the formula results are replaced with "#" characters.

To do this, select one of the following options:

- In the properties of the segments, select the options for automatic adjustment of the column width and row height.
- Click "Update all" in the report template.
  Values are imported to Excel from the data source. Check the column widths and row heights and adjust them manually, if required.

  Note

  The manual adaptations apply only in the Excel add-in. They are not included in the generated reports.

Replacing numerical values

If columns of a data source item output numeric values, you can assign text lists and graphic lists to the columns. When the Runtime data is read in, the cell values of these columns are replaced by texts or graphics from the assigned lists. This improves the readability of the report and helps to draw the reader's attention to important information.

Example: Visualizing limit violations of tags with graphics

See section Assigning text lists and graphic lists (Page 7708).

Preparing imported Runtime data

Adjust the cell formatting of the Runtime data, for example, font, color, alignment, or number format. The rows inserted when reading the Runtime data adopt the formatting.

Add diagrams, pivot tables or formulas that graphically visualize, structure or evaluate the data imported from Runtime.

Note

If you have read Runtime data into the report template for better data preparation, remove it before saving the report template and making it available for upload to Runtime.

To do this, click the "Delete Runtime data" button in the toolbar of the Excel add-in.

Set up page

Use "File > Print > Set up page" to define details for printing the report, for example:

- Alignment of the report (portrait format or landscape format)
- Scaling, for example, to print all columns on one page
- Inserting a user-defined header or footer

The print settings set in the report template are applied in Runtime when a report job is executed for PDF generation.

Renaming worksheets and segments

When you add a segment to a report template in the add-in, a table is created in Excel. The add-in addresses the table by the name of the worksheet and segment.

Do not rename worksheets after adding segments.

Do not change the table name of a segment using the Excel property "Table name". Edit the segment in the add-in and rename it there.

19.2.5 Working with production reports in Runtime

19.2.5.1 Workflow for working with reports in Runtime

Introduction

The following workflow describes which works are required in the "Reports" control so that production reports are generated in Runtime.

The reports can be stored as file in the file system and sent as an attachment to an e-mail. Alternatively, an e-mail without attachment can also be sent about the generation of the report. In this way, employees from management and production can be informed about the production situation promptly, regardless of their location.

You can send the e-mail using a secure SMTP server (authentication with user name and password or via certificate) or an unsecured SMTP server, for example, an internal company mail server.

Requirement

- Requirements in TIA Portal:
  - The necessary project data were configured for the HMI device for which reports are to be created.
  - The "Reports" control was placed on an HMI screen of the device.
  - The "Enable Reporting" option was enabled in the Runtime settings of the device.
  - (Optional) The storage locations for reports and the Reporting database were configured in the Runtime settings of the device.
- The HMI device has been compiled, uploaded to the Runtime server and its project is running.
- When using contexts: Contexts have been defined and executed in Runtime for the project.
- The Runtime server has access to report templates.
- For cross-project and cross-Runtime use of report templates: The data sources used in the report template can also be found on the HMI device. Make sure that the names and plant hierarchy are consistent.

Procedure

1. To send reports by e-mail, configure the global e-mail settings:
   - When one of the servers requires a certificate for sending e-mails, upload the certificate.
   - Create contacts for the e-mail receivers and e-mail senders.
   - Create the required SMTP server configurations.
2. Configure job parameters for report templates, triggers and targets.
   These job parameters will then be available to you for selection when configuring the report jobs.
3. Configure report jobs.
   Reports are generated in Runtime when the report jobs are executed.
4. (Optional) Perform report orders manually.
5. In the control, get an overview of which reports have been generated.
6. Download the reports, if necessary.
7. (Optional) To reuse the configuration of the "Reports" control, such as on a device in another network, transfer the existing configuration from the control from one device to the control of the other device.

Configuring job parameters

First, you configure which job parameters are available for selection during the configuration of the report jobs. You configure the following job parameters:

- The available report templates
  The report template defines which data the report outputs. Import and/or delete templates, if required.
- The available triggers
  The trigger defines when a report job is executed. Add triggers, edit triggers or delete them.
- The available targets
  Targets define whether reports are made available to users in the file system or via e-mails. Add targets, edit triggers, or delete them.

You set further job parameters while configuring a report job in the "Report jobs" tab.

Configuring a report job

You configure the following for each report job:

- Name of the report job
- Used report template
- Name of the reports generated by this template

Note

Texts through dynamic placeholders

Placeholders are available to you when defining the report name. The placeholders are evaluated and replaced by text during execution of the report.

See also Dynamic placeholder (Page 7746).

- Targets of the generated report
  To send e-mails, select a target of the type "E-mail".
- Per target: The target format of the generated report
  Possible formats: .XLSX and .PDF
- Trigger
- Comment
- Activate

See also

Setting global email settings (Page 7725)

Configuring task parameters (Page 7727)

Configuring report tasks (Page 7734)

Running a report job manually (Page 7742)

Downloading reports (Page 7742)

Transferring the control configuration (Page 7744)

Complete workflow for using production reports (Page 7652)

Creating report templates for production reports (Page 7655)

19.2.5.2 The user interface of the "Reports" control

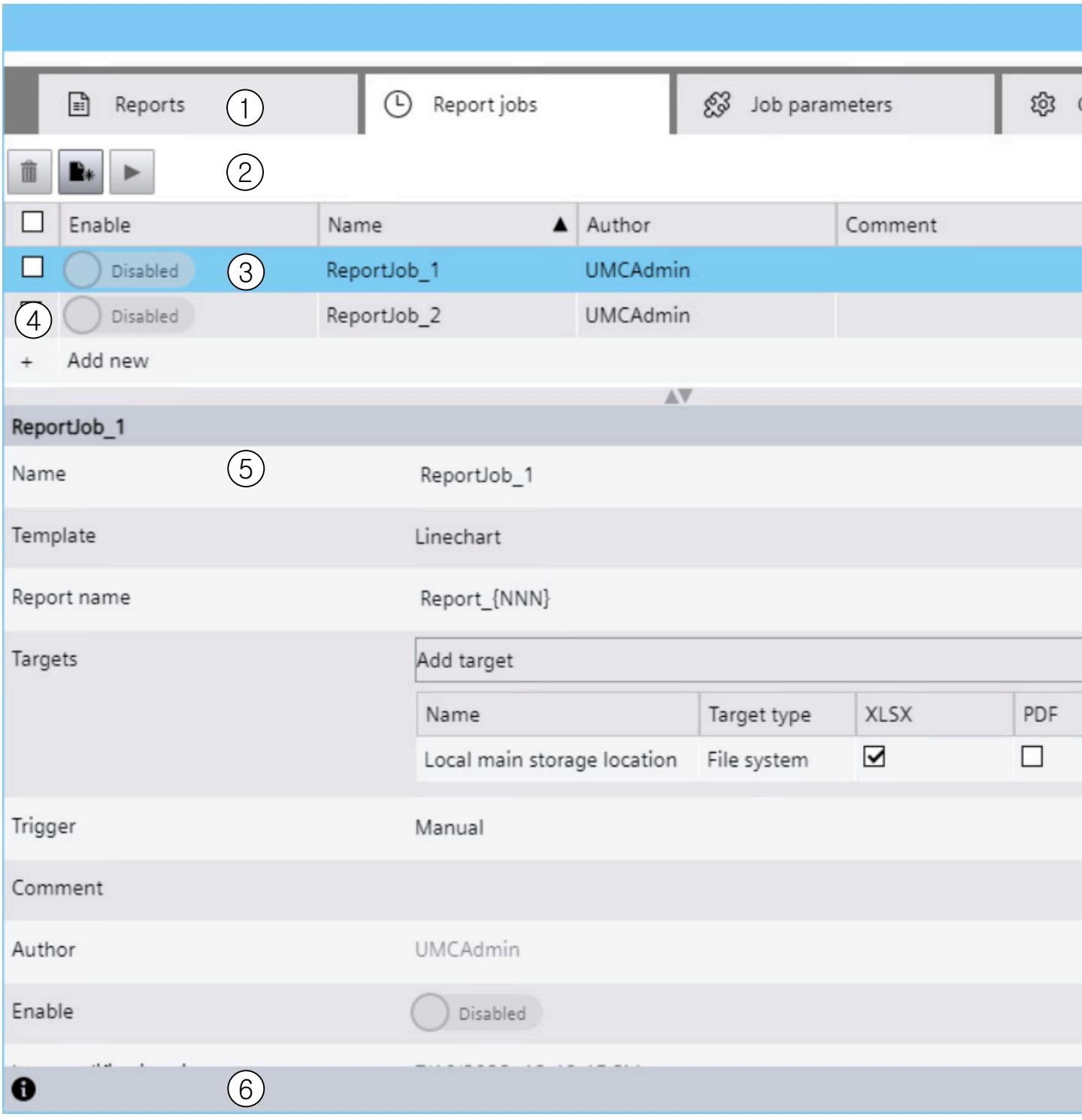
Note
Automatic data transfer

Changes in the "Reports" control are saved automatically.

Layout

You create and manage report jobs in the "Reports" control. You also have access to the reports generated by the report jobs.

The control has the following structure:



1 Tab for the configuration and management of reports, report jobs, job parameters and global settings

2 Toolbar
The buttons you see depend on the tab.

3 Work area
On the "Reports", "Report jobs" and "Job parameters" tabs: List of elements available on the tab
On the "Global settings" tab: The settings

4 Options for selecting the elements
You can select elements individually or all at once.

5 Detail area
Shows the properties of the selected element.

6 Information bar

Tab

"Reports" tab

Here you can see which reports have been generated. You can download or delete reports via the toolbar.

The "Status" column shows Information:

- On the status of the generated report files (XLSX and PDF)
- On the status of the targets (File system and E-mail)

Overview of the status icons:

Status	Description
	Execution has been successfully completed.
	An error occurred during execution.
	Execution is in progress.

A click on an icon opens a detailed status message.

"Report jobs" tab

Here you create new report jobs, manage existing report jobs or start a report job manually.

"Job parameters" tab

Here you manage the parameters with which you configure the report jobs in the "Report jobs" tab.

"Global settings" tab

Here you make the following settings:

- For sending e-mails
- For transfer of the control configuration
- For creating an offline configuration file
- For configuring paging

Toolbar

The following buttons are available in the toolbars of the tab:

Icon	Button	
	Delete	Deletes the elements whose option is enabled in the work area.
	• Add new • Import	• Creates a new element. • "Job parameters > Templates" tab: To import a report template into Runtime
	Run	In the "Report jobs" tab. Manually creates a report for the report job whose option is enabled in the work area.
	Export	• In the "Job parameters > Templates" tab: To export report templates • In the "Reports" tab: To download reports to the client

Information bar

The button in the information bar displays general information sent by the reporting service, for example, on whether a report job has been executed.

19.2.5.3 Setting global email settings

If configured accordingly, an e-mail is sent automatically after a report job is executed. The e-mail can include the report as an attachment.

Maintenance of the basic settings for sending e-mails is carried out in the "Global settings" tab:

- If necessary: The certificates that the e-mail sender uses to authenticate itself at the SMTP servers.
- The contact information of the e-mail senders and e-mail receivers.
- The configuration of the SMTP server via which the e-mails are sent.

Upload certificates

Store the certificates of the SMTP servers that require authentication via certificate.

Requirement

- You have access to the storage location of a valid certificate file.

Procedure

1. In the "Reports" control, click on the "Global settings > Certificates" tab.
2. Click "Add new" in the toolbar.
   Alternative: In the work area, click "Add new".
3. In the dialog that opens, select the certificate file.
4. Confirm your input.
5. Optional: Select the uploaded certificate in the work area and enter a comment on the certificate in the detail area.

Result

The certificates uploaded here are available in the "Contacts" tab.

Maintaining contacts

Store the data of the e-mail senders and email recipients.

Procedure

To create a new contact, follow these steps:

1. In the "Reports" control, click on the "Global settings > Contacts" tab.
2. Click "Add new".
3. Enter the name of the contact.
4. Enter the e-mail address of the contact.
5. To use the contact as a sender for an SMTP server that requires authentication with a certificate, select the appropriate certificate under "Certificate".
6. To use the contact as a sender for an SMTP server that requires authentication with a user name and password, enter the password.
   The e-mail address is used as the user name.
7. (Optional) Enter a comment relating to the contact.

Result

The contacts configured here are available:

- As the e-mail sender in the SMTP server configuration.
- As an e-mail recipient when configuring "target" job parameters with the target type e-mail

Maintenance of the SMTP server configuration

Store the data of the SMTP servers via which the e-mails are sent.

Requirement

Contacts that are suitable as senders have been entered in the "Global Settings > Contacts" tab.

Procedure

To create a new SMTP server configuration, follow these steps:

1. In the "Reports" control, click on the "Global settings > SMTP" tab.
2. Click "Add new".
3. Specify the following:

Field	Description
"Name"	Enter the name of the SMTP server configuration.
"Address"	Enter the URL of the SMTP server. Servers without authentication (e.g. company-internal mail servers) and with authentication are permitted. Example: URL of a company mail server: `mail.<Company name>.com`
"Port"	Enter the port number of the SMTP server. Default setting: 25
"Sender"	In the list, select the contact that is used as the sender for this SMTP server configuration. All contacts maintained under "Contacts" are offered to you for selection. Select a sender that meets the respective requirements of the server.
"Comment"	(Optional) Enter a comment relating to the SMTP server configuration.

Result

The servers configured here are available when configuring the "Target" job parameters with the target type email.

19.2.5.4 Configuring task parameters

Job parameters define the details of a report job.

You configure the following parameters on the "Job parameters" tab:

- Templates
- Trigger
  Define trigger when a report job is executed.
- Targets
  Targets define how a report is made available to users. The following target types are available:
  - "E-mail"
    An e-mail is sent after a report job is executed. The report generated by the report job can be included with the e-mail as an attachment.
  - "File system"
    The reports generated by the report job are stored in the file system.

The parameters configured here are available to you for selection when configuring the report jobs in the "Report jobs" tab.

You define the remaining job parameters while configuring a report job in the "Report jobs" tab.

See also

Importing and exporting report templates (Page 7728)

Deleting templates (Page 7729)

Configure trigger (Page 7729)

Importing and exporting report templates

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Templates" tab is visible in the control.
- Import: You have access to the storage location of the report template.
- Export: Report templates have been imported into the control.

Importing report template

1. Click "Add new" in the toolbar.
   Alternative: In the work area, click "Add new".
2. In the dialog that opens, select the file of a report template.

3. Confirm your input.

Note

No validation

The template is not validated during import.

4. Optional: In the work area, select the imported report template in the work area and enter a comment describing the template in the detail area.

Exporting report templates

1. In the work area, select the options next to the report templates you want to export.
2. Click "Export" in the toolbar.

The report templates are downloaded to the download folder or a user-defined directory according to the device settings.

Deleting templates

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Templates" tab is visible in the control.
- Templates have been imported into the control.

Procedure

1. In the work area, select the options next to the templates you want to delete.
2. Click "Delete" in the toolbar.

Deleting used templates

The "In use" column shows whether the template is used by a report job.

If you delete a template that is used by a report job, the report job is marked as inconsistent and no longer executed.

Configure trigger

Introduction

In the "Job Parameters > Triggers" tab you configure which automatic triggers are available for selection when configuring report jobs.

Report jobs with automatic triggers are executed if the report jobs on the "Report jobs" tab are set to active and their trigger event occurs. Users can also start the execution manually.

Requirement

- The "Reports" control is placed on a screen of the project.
- The "Job parameters > Trigger" tab is visible in the control.
- To use the trigger type "Context trigger": Contexts are available in the project.

Add trigger

1. In the work area of the tab, click "Add new".
   A new trigger is created and displayed in the detail area.
2. Assign a unique name to the trigger.
3. Select the trigger mode:

Trigger type	Triggering the trigger
"Tag trigger"	Automatically when the configured value condition occurs at the tag defined in the trigger.
"Serial trigger"	Automatically within the user-defined interval when the time defined by the series has been reached.
"Context trigger"	Automatically when the selected context is started or stopped. Optional: By using a condition, you can also limit the triggering of the trigger to specific context values.

4. Depending on the selected trigger type, set the settings for the new trigger as described below.
5. Optional: Enter a comment for the trigger.

Settings for tag trigger

1. Click "Select tag".
2. Click "Load".
3. Select the required tag and click "OK".
4. Set the condition and the condition value.
   Example:

Set tag	<tag name>
Condition	>
Condition value	100

The trigger will be initiated when the tag receives a value greater than 100.

Settings for serial triggers

1. Select the serial pattern.
   The series pattern defines the occurrence and time at which the trigger is initiated.
   Example: Weekly > Every 2 weeks > Fridays
2. Select the series area.
   The series range defines the period in which the trigger is initiated.

"Start"	Specify the start date
"Time"	Specify the time at which the trigger is initiated.
"End on"	Specify the end date. The trigger will be executed for the last time on this day.
"End after"	Determine the number of dates after which the series ends.
"No end date"	The series runs indefinitely.

Settings for context triggers

1. Click "Select context".
2. In the "Context selection" dialog, click "Select plant object".
3. In the "Browser view" dialog, select a plant object and confirm your input.
   In the "Context selection" dialog you can see all contexts that have been defined for the selected plant object.
4. Select a context and confirm your input.
5. Under "Context status", select when the trigger will be triggered:

"Started"	When starting the context.
"Stopped"	When stopping the context.

6. Optional: To bind the execution of the report order to certain context values, you define a condition:

"Condition"	Select an operator.
"Value"	Select a context value.

Example:

Plant object	"MyPlant.hierarchy::PlantView/Bottling"
Context	"Product"
Context state	"Started"
Condition	=
Value	"Orange lemonade"

Report jobs with this trigger are always executed when the context "Product" defined on the plant object "Bottling" is started with the value "Orange lemonade".

Delete trigger

Select the option of the desired trigger in the work area of the "Job Parameters > Triggers" tab and click "Delete" in the toolbar.

Edit trigger

1. Enable the option of the desired trigger in the work area of the tab.
2. In the detail area, edit the settings of the trigger.

Note
No change of the trigger type

The trigger type can only be set when adding the trigger.

Add target with target type "E-mail"

Requirement

- The "Reports" control is placed on a screen of the project.
- The receivers of the e-mails are maintained as contacts in the "Global settings > Contacts" tab.
- An SMTP server, with which the e-mail is to be sent, has been configured in the "Global settings > SMTP" tab.

Procedure

1. In the "Reports" control, click on the "Job parameters > Targets" tab.
2. In the work area of the tab, click "Add new".
3. Select "E-mail" as target type.
   A new target is created and displayed in the detail area.
4. Assign a unique name to the target.
5. Select an SMTP server configuration.
6. Add the desired receivers and CC receivers:
   – To do so, select a contact from the list "Add receiver" or "Add CC receiver".
   – Add the contact by clicking "+".
7. Enter the e-mail subject.
   To integrate the report name into the subject line, use the placeholder {ReportName}.
8. Enter the e-mail text.
   To integrate the report name into the email text, use the placeholder {ReportName}.
9. (Optional) Enter a comment.

Result

The target is available for selection when configuring report jobs.

An e-mail is sent after a report job is executed with this target. The e-mail can include the report as attachment.

See also

Dynamic placeholder (Page 7746)

Add a target with "File system" target type

Introduction

A reporting job with a target of the "File system" target type saves reports to a file system.

When configuring the report jobs, you can choose from pre-configured and user-defined targets of this target type.

Preconfigured targets

The following targets with "File system" target type are pre-configured:

Local project storage location	The reports are stored in the following folder: `<Project folder of the Runtime project>\Reports`
Local main storage location	The reports are stored in the local main storage location for reports. The local main storage location is configured in TIA Portal in the Runtime settings of the HMI device. If this setting has not been set in TIA Portal, the reports are stored as follows: • Unified PC: In the folder configured during installation of Runtime or later in the "WinCC Unified Configuration" tool • Unified Comfort Panel: In the "Reports" folder on the SD card inserted in the Panel: `media/simatic/X51/Reports`

You can select these targets in the "Report jobs" tab. You cannot edit or delete these targets in the "Job parameters > Targets" tab.

User-defined targets

In the "Reports" control, you can create user-defined targets of the "File system" target type. These user-defined targets are always subfolders of the local main storage location.

Requirement

- The "Reports" control is placed on a screen of the project.
- Unified Comfort Panel: The panel contains the storage media configured in the TIA Portal Runtime settings as storage locations for reports and for the reporting database.

Procedure

To add user-defined targets of the "File system" target type, follow these steps:

1. In the "Reports" control, click on the "Job parameters > Targets" tab.
2. In the work area of the tab, click "Add new".

3. Select "File system" as target type.
   A new target is created and displayed in the detail area.
   Under "Destination path", you can see the path to the local main storage location for reports.
4. Assign a unique name to the target.
5. Under "Subfolder", enter the path to the subfolder in which the report is to be saved.
   Use the following notation: <folder name> or <folder name>\<folder name>\...

   Note
   Relative path information

   The path specification is relative to the local main storage location for reports.

6. (Optional) Enter a comment.

Result

The target is available for selection when configuring report jobs.

When a report job with this target is being executed, the generated report is stored in the subfolder of the local main storage location defined as the target. If the folder entered under "Target path" does not exist, it is created by the system.

Note
Change of the local main storage location for reports

When the local main storage location for reports changes, the targets are automatically adapted. New reports are stored relative to the new local main storage location. The old folders are not deleted.

19.2.5.5 Configuring report tasks

Creating a report job

Introduction

A *report job* is a job for generating reports in Runtime. The configuration of a report job controls the details of the generation.

Requirement

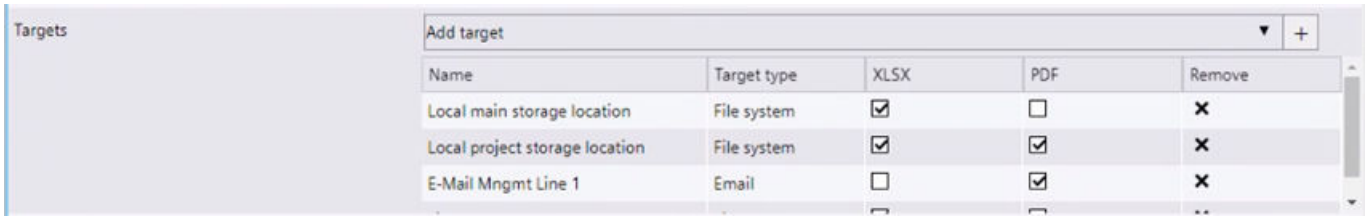
- The "Reports" control is configured on a screen of the project.
- The following job parameters were configured in the control:
  - At least one template has been imported.
  - To automatically execute a report job: Triggers are configured in the "Job parameters > Trigger" tab.

- For sending an email after execution of the report job:
  - Email contacts were configured in the global settings.
  - An SMTP server was configured in the global settings.
  - A target of the target type "E-mail" was configured in the "Job parameters > Targets" tab.
- For a report job with the target format PDF:
  - Microsoft Office Excel or LibreOffice is installed on the runtime server.
  - Depending on whether Excel or LibreOffice is installed, the information required for PDF creation was provided during the Runtime installation or in the "WinCC Unified Configuration" tool.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. Select "Add new" in the work area or click "Add new" in the toolbar.
3. In the detail area, enter a name for the report job.
4. Select a report template.
5. Configure the report name. See section Configuring report names (Page 7737).
   The configuration is applied to all reports generated by the report job.

6. Under "Targets", you determine how the reports are to be made available to users. Follow these steps:
   - Click "Add target".
     You see the targets configured in the tab "Job parameters > Targets".
   - Select a target.
   - Add the target by clicking "+".
     The target is added to the table to define the target formats.



   - Determine the formats in which the reports generated by the report job are provided for the target. In the table, activate the options of the desired formats for each target.

     Note

     Sending emails without a report

     If you deactivate both options for targets with "E-mail" target type, an email without attachment is sent after the report job has been executed.

     Note

     PDF as target type

     Generating PDFs with Excel is significantly slower than with LibreOffice. To generate large PDF reports, it is therefore recommended that you install LibreOffice.

     A PDF report created by LibreOffice can deviate in content or layout from a PDF report generated with Excel, for example, if the report template uses common Excel features that LibreOffice does not support, such as special fonts or chart types.

   - To remove a target from the report job, click the "Remove" button in the table.
7. Under "Trigger", select which event triggers the execution of the report job:
   - If the report job is only to be executed manually, select "Manual".
   - If the report job is to be executed automatically, select one of the other triggers configured under "Trigger".

     Note

     You can also execute the report job manually.

8. (Optional) Enter a comment for the report job.
9. Specify whether the automatic execution of the report job is active or paused. To do this, set the slider "Enabled" or "Disabled".

   Note

   You can still execute disabled report jobs manually.

Result

The report job is saved automatically.

When its trigger occurs, the report job is executed. A report is generated and made available as configured under "Targets".

See also

Execution of report jobs (Page 7741)

Configure trigger (Page 7729)

Add target with target type "E-mail" (Page 7732)

Add a target with "File system" target type (Page 7733)

Tips on design and layout (Page 7717)

Managing report jobs

Requirement

- The "Reports" control is configured on a screen of the project.
- Report jobs have been configured in the control.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. To edit a report job, proceed as follows:
   - Select the report job in the work area.
   - In the detail area, edit the settings of the report job.
     You have the same options as when creating a report job.
3. To delete report jobs, proceed as follows:
   - In the work area, enable the options next to the report job.
   - Click "Delete" in the toolbar.

Configuring report names

Note

Make sure that the generated report name does not violate the policy of the operating system regarding the maximum length of file names.

Introduction

The default name of reports is `Report_{NNN}`.

To use different report names, enter one or more placeholders at the report job. The placeholders are combined to form the report name during execution of the report.

Placeholder types

Placeholders have one of the following types:

Placeholder type	Description	
Text	For user-defined fixed texts	
Counter	On automatic numbering	Dynamic placeholders The placeholders are broken down into values during execution of the report.
Date format	For outputting the generation time	
Tag	To output the process value of an online tag	

Unique report names

If the report name uses counter or date format placeholders, the report job generates unique report names.

Requirement

- The "Reports" control contains a screen of the runtime project that is running.

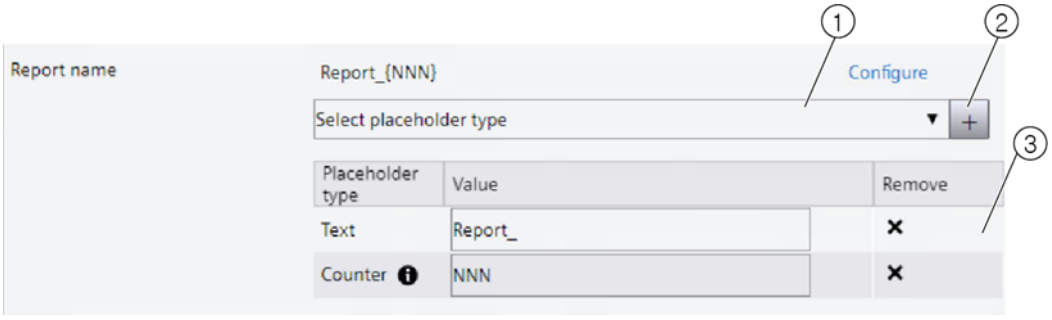
Procedure

You can enter the placeholders manually in the "Report name" field or you can have the software help you configure the report name.

To have the software help configure the report name, follow these steps:

1. Select the "Report jobs" tab in the "Reports" control.
2. Select a report job in the work area.
   You can see the settings for the report job in the detail area.

3. Next to "Report name", click "Configure".
   You see the following operator controls:



① List for selection of the placeholder type
② Button for adding a placeholder of the selected type
③ Table for configuring or removing the placeholder

Note

For the default report name, the "Report name" has the value `Report_{NNN}` and the table shows the placeholders "`Report_`" and "`NNN`".

To swap the order of placeholders or to add a placeholder in the center, delete the placeholders and then add them in the desired order.

4. Optional: To delete the default placeholders, click "x" in the placeholder table.
5. Select the desired type under "Select placeholder type".

Note

A report name can contain only one counter.

6. Click "+".
   An empty placeholder of the corresponding type is added at the end of the table.
7. Enter the placeholder under "Value" in the placeholder table.

Placeholder type	Description	Example
"Text"	Enter the text.	`Report_`
"Date format"	Enter a date placeholder.	A list of permitted placeholders and examples can be found in section Dynamic placeholder (Page 7746).
"Counter"	Enter a counter placeholder.	
"Tag"	Enter the full name of an online tag.	`RT1_Brewery::BatchNo`

Note

Enter the dynamic placeholders without any markup characters.

Alternatively, you can select an online tag via the user interface. Follow these steps:

– Click the "..." button on the tag placeholder.

– In the "Tag selection" dialog, click the "Search" button.
  You can see all the tags of the Runtime project that is running.

  Note

  Scrolling and filtering

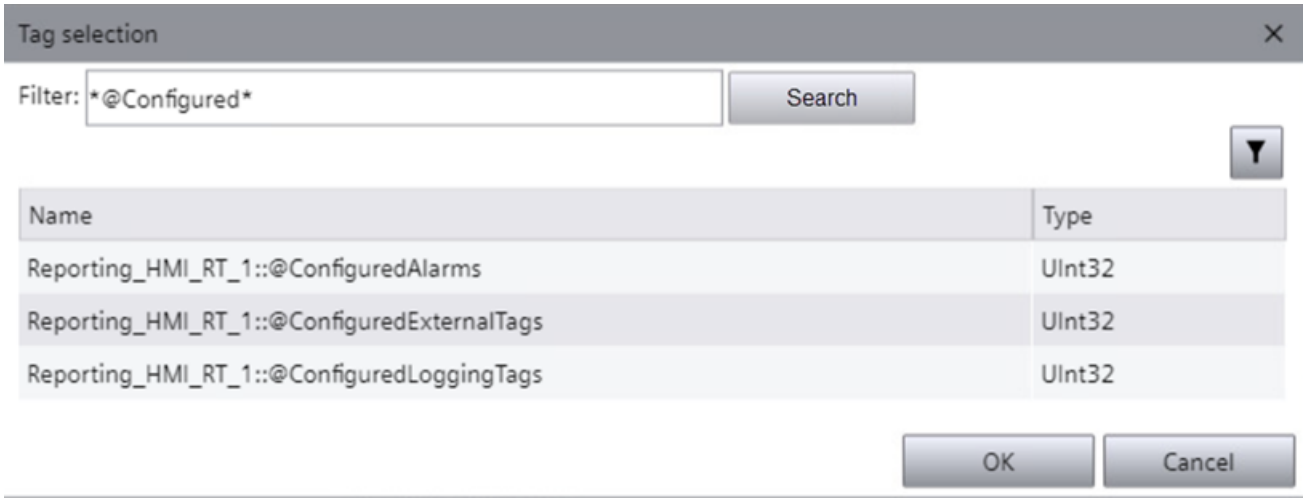
  Use the page navigation buttons to scroll forward or backward.

  To filter the displayed tags, enter a filter string in "Filter" and click "Search".

  You use the wildcard "*" to filter by partial strings.

  For example:
  - *T* returns all tags with a "T" in their name.
  - *T returns all tags that end in "T".
  - T* returns all tags that start with "T".

  When filtering for structures, the separators must be part of the filter string.



– Click the desired tag.

– Confirm with "OK".

In the "Report name" field, the placeholder you added is appended to the end of the report name.

Alternative procedure

To enter the placeholders manually, proceed as follows:

1. Select the "Report jobs" tab in the "Reports" control.
2. Select a report job in the work area.
   You can see the settings for the report job in the detail area.
3. Enter the desired combination of fixed texts and dynamic placeholders manually in the "Report name" field.
   Use markup characters for the dynamic placeholders. See section Dynamic placeholder (Page 7746).

Example:

"Report name" value	Generated report name
`Report_{yyyymmdd}_{HHMMss}_{@PC1_Brewery::BatchNo}`	`Report_20190101_170001_BatchNo_87002314`

Result

When generating a report, the dynamic placeholders are resolved and all placeholders are merged to form the report name.

If a process value contains a character that is not permitted in file names, it is replaced by an underscore.

If there is an error resolving the name, e.g. because the tag is not found in runtime, the tag placeholder in the name is replaced by `ERR`. The process is logged in the generation status of the report.

Execution of report jobs

Automatic and manual execution

Automatic execution

Report jobs that have a tag trigger, serial trigger or context trigger and are set to active on the "Report jobs" tab are automatically executed when their trigger occurs.

Manual execution

Report jobs with a trigger of the "Manual" type must always be executed manually.

In addition, you can at any time manually execute report jobs that have a tag trigger, serial trigger or context trigger.

System response to errors

- Error adding the report job to the queue
  The execution of the report job is discarded. A system alarm documents the error.
- Error executing the job
  In the "Reports" control, "Reports" tab, the status icon indicates the error. A click on the icon opens a detailed status message.
  A system alarm documents the error.

See also

Running a report job manually (Page 7742)

Configure trigger (Page 7729)

19.2.5.6 Running a report job manually

You can execute report jobs manually at any time, regardless of their trigger type. This also applies to report jobs that were disabled in the "Report Jobs" tab and whose automatic execution is therefore paused.

Requirement

Report jobs have been configured in the "Reports" control.

Procedure

1. Select the "Report jobs" tab in the "Reports" control.
2. In the work area, enable the option next to the report job that you want to execute manually.
3. Click "Execute" in the toolbar.

Result

The report is generated. You can download it in the "Reports" tab.

19.2.5.7 Downloading reports

You can download the reports stored by the report job in the file system to your device.

Depending on which file formats have been set in the report job, you can download the report as an XLSX file and as a PDF file.

Requirement

- Report jobs with the target type "File system" have been configured and executed in the "Reports" control.

Procedure

1. Select the "Reports" tab in the "Reports" control.
2. In the work area, select the option in the left column for each report that you want to download.
3. Enable the desired target formats in the "Files" column.

   Note

   Generation status

   You are only offered successfully generated formats.

   In the "Status" column you can check whether the generation for a format has failed. For a detailed status message click on the icon of a target format.

4. Click "Export" in the toolbar.

Result

The reports are downloaded into the download directory of the browser.

You can edit, distribute, or log the reports.

19.2.5.8 Exporting an offline configuration file

An offline configuration file is required to configure reporting templates in the Reporting Excel add-in without an online connection to the Runtime server.

Requirement

- The "Reports" control is placed on a screen of the project.
- The Runtime project has data that can serve as data source elements in the reporting template, such as alarms and logging tags.

Procedure

1. In the "Reports" control, click on the "Global settings > Configuration" tab.
2. Enter the name of the offline configuration file under "Offline-configuration".
3. Click "Export offline configuration".

Result

A JSON file with the data source elements of the Runtime project is created. The file is downloaded to the download folder or a user-defined directory according to the device settings.

You can select the configuration file in the Reporting Excel add-in as data source for an offline connection.

19.2.5.9 Transferring the control configuration

You have the option of reusing the settings in the "Reports" control, for example, on a device in another network. To do this, export the existing configuration on the one device from the control to a ZIP file. Then import the file into a "Reports" control on the other device.

Scope

The transfer covers the following data:

- Global settings, without passwords and certificates
- Job parameters, including the report templates available in the control
- Report jobs

Reports are not transferred.

Requirement

- The "Reports" control is placed on a screen in the project running in Runtime.
- Export: Settings have been made, e.g. contacts maintained, report templates imported, and report jobs created in the "Reports" control.
- Import: You have access to the ZIP file generated by the export on the device on which Runtime is installed.

Export configuration

1. In the "Reports" control, select the "Global settings > Configuration" tab.
2. Enter the name of the export file under "Export/import configuration > Export".
3. Click "Export configuration".

The configuration is exported to a ZIP file and downloaded to the default download directory of the device.

Import configuration

1. In the "Reports" control, select the "Global settings > Configuration" tab.
2. Click "Select import file" under "Export/import configuration".
3. Select the ZIP file in File Explorer and confirm your selection.
4. Runtime checks whether the control already contains configurations:
   - No: The configuration is imported.
   - Yes:
     Select "OK" to import the configuration. The existing configuration is overwritten.
     Select "Cancel" to cancel the import.

19.2.5.10 Configuring enable paging

To set how many entries the lists in the work area of the "Reports" control display per page, follow these steps:

1. In the "Reports" control, click on the "Global settings > Configuration" tab.
2. Under "List Settings", select the number of entries.

If a list has more entries, these are split over several pages. Use the buttons below the list to switch pages.

Note

The setting is lost through a screen change.

19.2.5.11 Inconsistencies and error diagnostics

Note

Inconsistent report jobs are not executed.

The templates available in the "Reports" control are not validated.

Display of inconsistencies and errors

Errors and inconsistencies are displayed as follows:

In the control	If job parameters are affected. Examples: • No template is set for a report job. • A tag that triggers a report job is deleted in the engineering system. The project is reloaded into the device.
In the "Error log" worksheet of the report	Errors or inconsistencies affecting the content of the report. Example: The report evaluates data from a tag that is no longer available in runtime.
As system alarm	For errors and inconsistencies that do not affect job parameters or the contents of the report. Example: The ExecuteReport system function transfers a report job that does not exist.

Job parameters

The following values lead to errors and inconsistencies:

Parameter	Invalid values	Default setting
"Name"	Zero, empty or already assigned name	"New report job"
"Template"	Zero, empty or "None". Name of a template that is not imported	"None"
"Target name"	Zero or empty	"NewReportJob[NN]"

19.2.5.12 Dynamic placeholder

Introduction

Dynamic placeholders are evaluated when the report job is executed and replaced with text in runtime.

The following job parameters can contain placeholders:

- Report name
- Targets with the target type "E-mail": Subject and text of the email

Dynamic placeholders for report names

Use dynamic placeholders for counters and/or dates to generate unique report names:

Counter placeholder	Description	Example		Area
		Configuration	Result	
{N}	Automatic numbering	Rep_{N}	Rep_1	0...9
{NN}		Rep_{NN}	Rep_01	00...99
{NNN}		Rep_{NNN}	Rep_001	000...999

Date placeholder	Description	Example		Area
		Configuration	Result	
{yy}	Current year	Rep_{yy}	Rep_18	Valid year with 2 digits
{yyyy}		Rep_{yyyy}	Rep_2018	Valid year with 4 digits
{m}	Current month	Rep_{m}	Rep_1	Valid month, no prefixed 0 for months in single-digit range
{mm}		Rep_{mm}	Rep_01	Valid month, prefixed 0 for months in single-digit range
{mmm}		Rep_{mm}	Rep_Jan	Month abbreviation with 3 characters
{mmmm}		Rep_{mmmm}	Rep_January	Month with full name

Date placeholder	Description	Example		Area
		Configuration	Result	
{d}	Current day of the month	Rep_{d}	Rep_1	Valid day, no prefixed 0 for days in single-digit range
{dd}		Rep_{dd}	Rep_01	Valid day, prefixed 0 for days in single-digit range
{ddd}		Rep_{ddd}	Rep_Mon	Day abbreviation with 3 characters
{dddd}		Rep_{dddd}	Rep_Monday	Day with full name
{h}	Current hour	Rep_{h}	Rep_1	Current hour (12-hour clock), no prefixed 0 for single-digit values
{hh}		Rep_{hh}	Rep_01	Current hour (12-hour clock), prefixed by 0 for single-digit values
{H}		Rep_{H}	Rep_13	Current hour (24-hour clock), no prefixed 0 for single-digit values
{HH}		Rep_{HH}	Rep_13	Current hour (24-hour clock), prefixed by 0 for single-digit values
{M}	Current minute	Rep_{M}	Rep_6	Valid minute, no prefixed 0 for single-digit values
{MM}		Rep_{MM}	Rep_06	Valid minute, prefixed by 0 for single-digit values
{s}	Current second	Rep_{s}	Rep_41	Valid second, no prefixed 0 for single-digit values
{ss}		Rep_{ss}	Rep_41	Valid second, prefixed by 0 for single-digit values

Use a dynamic placeholder for tags to integrate process values in the report name:

Tag placeholder	Description	Example		Area
		Configuration	Result	
{@<Full Tag name>}	Process value of an online tag	Rep_{@PC1_LineA::MyTag1}	Rep_On	Process value of the online tags If the value contains a character that is not permitted in file names, it is replaced by an underscore. If there is an error resolving the name, e.g. because the tag is not found in runtime, the tag placeholder in the name is replaced by `ERR`. The process is logged in the generation status of the report.

Examples:

Definition with placeholder	Generated report name
LineA_{yyyymmdd}_{HHMMss}	LineA_20190101_170001
LineA_{yymmmd}_{hhMMss}	LineA_19Jan1_050001
LineA_{NNN}	LineA_014
LineA_{yyyymmdd}_{HHMMss}_BatchNo_{@PC1_Brewery::BatchNo}	LineA_20190101_170001_BatchNo_87002314

Placeholder for email subject and email text

To integrate the report name into the subject line or the email text, use the following dynamic placeholder {ReportName}.

Markup

Use the following markup characters for dynamic placeholders:

- Placeholders for counter and date: `{ }`
- Placeholders for tags: `{@}`

Note

There is no markup in the placeholder table for defining the report name. See also section Configuring report names (Page 7737).

Runtime Openness 20

20.1 WinCC Unified Open Pipe

20.1.1 Introduction

Welcome to WinCC Unified Open Pipe

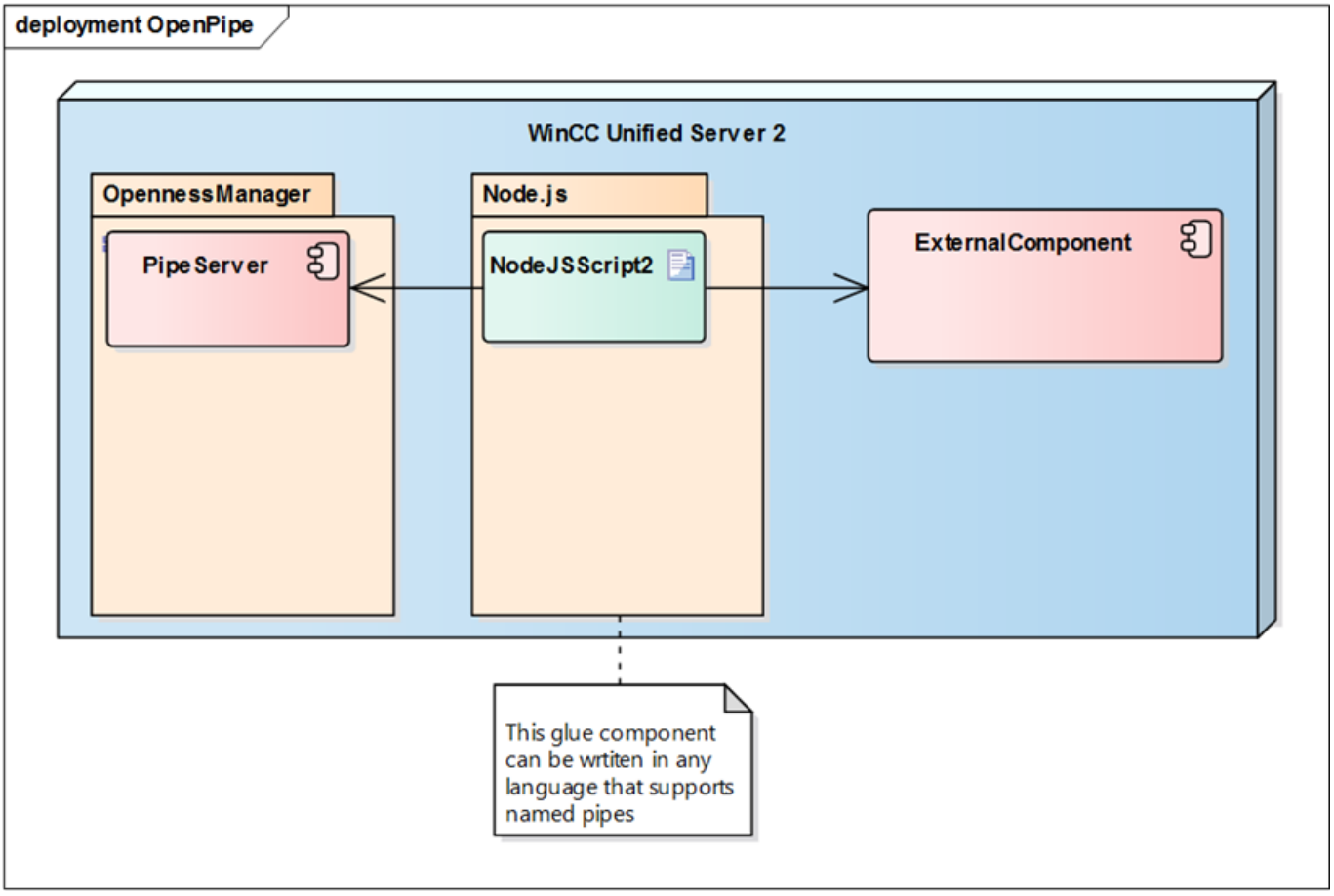
WinCC Unified Open Pipe is an Openness concept based on pipe technology to connect a customer application to WinCC Unified RT.

Compared to Openness RT (ODK), WinCC Unified Open Pipe offers a limited number of functions. However, the connection code can be written in any programming language that supports pipe technology. Even batch access to the pipe is possible.

The main aim of WinCC Unified Open Pipe is to make it possible for you to connect an existing application to WinCC Unified RT with little workload and independent of the programming language. The available commands let you communicate with WinCC Unified RT using tags and alarms.

Pipe technology

The pipe is a data stream with buffer between two processes that works according to the FIFO principle (First In First Out). One process is provided by WinCC Unified RT (OpennessManager). It creates the pipe and processes the requests of the customer application. The second process is the customer application. It connects to the pipe by using its name, sends requests and receives responses.



Name of the pipe:

- Under Windows: "\\.\pipe\HmiRuntime"
- Under Linux: "/tmp/HmiRuntime"

As soon as the pipe is open, single-line commands can be sent; they must end in a line break ("\n" or "\r\n"). Responses are returned using the same pipe instance.

Basic syntax and expert syntax

Two types of syntax are available for WinCC Unified Open Pipe:

- Basic syntax
  You use the basic syntax when you are working with basic batch files (e.g. in a CMD.exe or batch).
- Expert syntax
  You use the expert syntax when you are working with scripts or programming languages that have a JSON parser, e.g. Python, Node.js or Powershell.

Samples

You will find the following file on the installation medium:
"Support\Openness\Siemens.Unified.Openness_SDK_<version number>.zip"

Extract the file locally to any directory on your computer. You will then find examples of the use of WinCC Unified Open Pipe in the subfolder "OpenPipe\Samples".

20.1.2 Safety-related settings

Pipe access

- The use of Open Pipe is limited to local communication between the Open Pipe application and Runtime.
- To access the pipe, the user of the client script must belong to the following user group:
  - Under Windows: "SIMATIC HMI"
  - Under Linux: "industrial"

20.1.3 Behavior of the browse commands

Browse behavior

WinCC Unified Open Pipe provides several Browse commands in the simple syntax and in the expert syntax. The following behavior is common to all commands:

- Initial request
  In an initial request, you define the request using optional parameters.
- Page size
  To get the result faster, you can use a parameter to specify a page size in the initial request. If the result exceeds the specified page size, the response is distributed over several pages. Without this parameter the configured page size is used. You can read the configured page size via the commands "ReadConfig" and set it via "WriteConfig".

- Next request
  You get the next results page via the following requests. Next requests only have the parameter "Next".
- A Browse command is complete when a Next request returns an empty response.
- A Browse command that has not yet reached the last page of results is canceled in the following cases:
  - When the configured time limit for inactivity is exceeded.
    You can read the time limit via the commands ReadConfig and set it via WriteConfig.
  - When a new browse request is started.

Configurable settings

The following settings control the browse behavior:

Setting	Default value	Value range	Description
DefaultPageSize	1000	0 ... Uint32_Max	If a browse request was called without the PageSize parameter and the number of hits is greater than the value of DefaultPageSize, the response is distributed over several pages. Value 0: No pagination
BrowseTimeOut	300 s	0 ... Uint32_Max	For requests with pagination, defines after how many seconds without activity the request is canceled. After that a Next request returns an error. Value 0: No timeout

With the command "ReadConfig" you request the settings, with the command "WriteConfig" you set them. Changes apply to the current pipe session.

20.1.4 Using basic syntax

20.1.4.1 Basics of basic syntax

Characteristics

The basic syntax has the following characteristics:

- Simple text-based syntax without JSON parts.
- Only commands for individual objects, e.g. write access to a single tag.
- Object names may not include special characters or space characters.
- No cookies.

Structure of a request

`<Command> <Object> <Value>`

- `Command`: Command name, e.g. "WriteTagValue"
- `Object`: Name of the object for which the command is called, e.g. "Tag1"
- `Value`: Input value, e.g. "True"

Structure of a response

OnSuccess:

`Notify<Command> <Object> <Value1…ValueN>`

- `Command`: Command name, e.g. "ReadTagValue"
- `Object`: Name of the object for which the command was called, e.g. "Tag1"
- `Value1...ValueN`: For example, value and quality of the read tags

OnError:

`Error<Command> <Object> <Error text>`

- `Command`: Command name, e.g. "ReadTagValue"
- `Object`: Name of the object for which the command was called, e.g. "Tag1"
- `Error text`: Detailed error description

Overview of the commands of the simple syntax

Command	OnSuccess	OnError	Description
`SubscribeTagValue <Tag>`	`NotifySubscribeTagValue <Tag> <Quality> <Value>`	`ErrorSubscribeTagValue <Tag> <Error text>` `ErrorNotifyTagValue <Tag> <Error text>`	Subscribe tag for monitoring. `Quality`={good, bad, uncertain}
`UnsubscribeTagValue <Tag>`	`NotifyUnsubscribeTagValue <Tag>`	`ErrorUnsubscribeTagValue <Tag> <Error text>`	Unsubscribe tag from monitoring.
`ReadTagValue <Tag>`	`NotifyReadTagValue <Tag> <Quality> <Value>`	`ErrorReadTagValue <Tag> <Error text>`	Reads the value of the tags from the system.
`WriteTagValue <Tag> <Value>`	`NotifyWriteTagValue <Tag>`	`ErrorWriteTagValue <Tag> <Error text>`	Writes the value to the tag.
`SetCharSet <Value>`	`NotifySetCharSet <Value>`	`ErrorSetCharSet <Value> <Error text>`	Changes to a different character coding.
`ReadConfig <Parameter>`	`NotifyReadConfig <Parameter> <Value>`	`ErrorReadConfig <Error text>`	Reads a setting configured for the general browse behavior.
`WriteConfig <Parameter>`	`NotifyWriteConfig <Parameter> <Value>`	`ErrorWriteConfig <Error text>`	Sets a setting for the general browse behavior.

Command	OnSuccess	OnError	Description

`BrowseTags`:
• Initial request:
`BrowseTags <System> <Page size> --filter <Filter>`
• Next request:
`BrowseTags --next`
	`NotifyBrowseTags <System>::<Tag> ... <System>::<Tag>`	`ErrorBrowseTags <Error text>`	Returns the names of the tags of the local system or all systems communicating via runtime collaboration.

`BrowseConfiguredAlarms`:
• Initial request:
`BrowseConfiguredAlarms <System> <Page size> --filter <Filter>`
• Next request:
`BrowseConfiguredAlarms --next`
	`NotifyBrowseConfiguredAlarms <System>::<Alarm> ... <System>::<Alarm>`	`ErrorBrowseConfiguredAlarms <Error text>`	Returns the names of the configured alarms of the local system or all systems communicating via Runtime Collaboration.

`BrowseAlarmClasses <System>`	`NotifyBrowseAlarmClasses <System>::<Alarm class> ... <System>::<Alarm class>`	`ErrorBrowseConfiguredAlarms <Error text>`	Returns the alarm classes of the local system or all systems communicating via Runtime Collaboration.

Errors and error texts

This help only provides a selection of possible error messages. The error texts can also differ from the texts in your projects.

20.1.4.2 Commands

SubscribeTagValue

Description

The command "SubscribeTagValue" subscribes the specified tag for monitoring.

Error handling

- When the tag value contains a line break (\n), the value cannot be signaled. An error is signaled.
- When the same tag is subscribed a second time for monitoring, an error is signaled.

- A global monitoring error is signaled with "ErrorSubscribeTagValue". Because no monitoring was set up, there is no need to unsubscribe the tag from monitoring.
- An error relating to the tag value is signaled with "ErrorNotifyTagValue". Monitoring is set up in this case, but the tag value cannot be signaled for various reasons. Unsubscribe the tag from monitoring when it is no longer needed.

Request

```
SubscribeTagValue <Tag>
```

For example: `SubscribeTagValue Tag_1`

Response

OnSuccess (or partial success):

```
NotifySubscribeTagValue <Tag> <Quality> <Value>
```

For example:

- `NotifySubscribeTagValue Tag_1 Uncertain 0`
- `NotifySubscribeTagValue Tag_1 Good 10`
- `NotifySubscribeTagValue Tag_1 Bad 12`

OnError:

- Global error:
  `ErrorSubscribeTagValue <Tag> <Error text>`
  For example:
  - `ErrorSubscribeTagValue Tag_1 Tag does not exist`
  - `ErrorSubscribeTagValue Tag_1 Subscription already exists`
- Error for tag value:
  `ErrorNotifyTagValue <Tag> <Error text>`
  For example:
  - `ErrorNotifyTagValue Tag_1 Encoding error`
  - `ErrorNotifyTagValue Tag_1 Value contains newline`

UnsubscribeTagValue

Description

The "UnsubscribeTagValue" command unsubscribes a tag from monitoring.

Request

```
UnsubscribeTagValue <Tag>
```

For example: `UnsubscribeTagValue Tag_1`

Response

OnSuccess:

`NotifyUnsubscribeTagValue <Tag>`

For example:

- `NotifyUnsubscribeTagValue Tag_1`
- `NotifyUnsubscribeTagValue Tag_1`
- `NotifyUnsubscribeTagValue Tag_1`

OnError:

`ErrorUnsubscribeTagValue <Tag> <Error text>`

For example: `ErrorUnsubscribeTagValue Tag_1 Subscription does not exist`

ReadTagValue

Description

The "ReadTagValue" command reads the value of a tag from the system. Only the tag value and the quality are signaled.

When the tag value contains a line break (\n), the value cannot be signaled. An error is signaled.

Request

`ReadTagValue <Tag>`

For example: `ReadTagValue Tag_1`

Response

OnSuccess (or partial success):

`NotifyReadTagValue <Tag> <Quality> <Value>`

For example:

- `NotifyReadTagValue Tag_1 Uncertain 0`
- `NotifyReadTagValue Tag_1 Good 10`
- `NotifyReadTagValue Tag_1 Bad 12`

OnError:

`ErrorReadTagValue <Tag> <Error text>`

For example:

- `ErrorReadTagValue Tag_1 Tag does not exist`
- `ErrorReadTagValue Tag_1 Encoding error`
- `ErrorReadTagValue Tag_1 Value contains newline`

WriteTagValue

Description

The "WriteTagValue" command writes a value to a single tag.

When the transferred tag value contains a line break (\n), only the partial string in front of the line break is written to the tag.

Request

```
WriteTagValue <Tag> <Value>
```

For example: `WriteTagValue Motor.Label MC001`

Response

OnSuccess (or partial success):

```
NotifyWriteTagValue <Tag>
```

For example: `NotifyWriteTagValue Motor.Label`

OnError:

```
ErrorWriteTagValue <Tag> <Error text>
```

For example: `ErrorWriteTagValue Motor.Label Tag does not exist`

SetCharSet

Description

Sets the character encoding to one of the following specified values: {UTF-8, cp437, cp850}

The default character encoding is UTF-8.

The following character coding values must be supported as a minimum:

- UTF-8
- cp850
  In German Windows systems, this is the default for the SystemLocale.
- cp437
  In US Windows systems, this is the default for the SystemLocale.

Internally, strings are treated as Unicode strings (often as UTF-16 in a CFSTR). For external communication over the pipe, the Unicode characters must be converted into a byte representation.

When a character cannot be converted for a specific character coding (e.g. the Greek character "π" in the character coding cp437), an "encoding error" is triggered.

Request

```
SetCharSet <Value>
```

For example: `SetCharSet UTF-8`

Response

OnSuccess:

```
NotifySetCharSet <Value>
```

For example: `NotifySetCharSet UTF-8`

OnError:

```
ErrorSetCharSet <Value> <Error text>
```

For example: `ErrorSetCharSet UTF-9 unknown character set`

ReadConfig

Description

The "ReadConfig" command reads a setting configured for general browse behavior. To read multiple settings, call the command several times.

Request
`ReadConfig <Param>`

- `Param`:
  Value: The configuration parameters
  Possible parameters:
  - `DefaultPageSize`
    The page size used when a browse request is called without the "PageSize" parameter
  - `BrowseTimeOut`
    Number of seconds after which an inactive browse request is aborted.

For example:
`ReadConfig DefaultPageSize`

OnSuccess
`NotifyReadConfig <Parameter> <Value>`

For example:
```
NotifyReadConfig DefaultPageSize 1000
NotifyReadConfig BrowseTimeOut 300
```

OnError
`ErrorReadConfig <Error description>`

For example:
`ErrorReadConfig Invalid arguments passed to browsing function.`

WriteConfig

Description

The "WriteConfig" command sets a setting configured for general browse behavior. To set multiple settings, call the command several times

Request
`WriteConfig <Parameter> <Value>`

- `Parameter`:
  Value: The configuration parameters
  Possible parameters:
  - `DefaultPageSize`
    The page size used when a browse request is called without the "PageSize" parameter
    Preset value: 1000
  - `BrowseTimeOut`
    Number of seconds after which an inactive browse request is canceled.
    Preset value: 300 s

For example:
`WriteConfig DefaultPageSize 500`

Response

OnSuccess:
`NotifyWriteConfig <Parameter>`

For example:
`NotifyWriteConfig DefaultPageSize`

OnError:
`ErrorWriteConfig <Error description>`

For example:
`ErrorWriteConfig Invalid arguments passed to browsing function.`

BrowseTags

Description

The "BrowseTags" command returns the names of the tags of the local HMI system or all HMI systems communicating via runtime collaboration.

Information on the general browse behavior of the command can be found in section Behavior of the browse commands (Page 7751).

If the number of hits exceeds the page size of the response, the command consists of an initial request and 1 to N Next requests:

- You define the query in the initial request.
  It provides a response that delivers the first set of hits.
- You can call the remaining hits via Next requests.

Initial request

`BrowseTags <System> <Page size> --filter <Filter>`

- `System`: Optional
  Controls from which system the tags are read.
  - Value: "*"
    All systems that communicate with each other via runtime collaboration
  - Default value: The local system
- `Page size`: Optional
  Controls how many tags a response returns.
  Default value: The configured page size is used. See also section ReadConfig (Page 7758).
- `--filter <Filter>`: Optional
  Restricts the command to tags whose "Name" matches the filter.
  - `--filter`: Command line command
  - `<Filter>`: The filter string
    Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	Filter string: "Motor*" Tags contained in the response: • "Motor" • "MotorOn" • "MotorOff"
?	Replaces 1 character	Filter string: "Motor_?" Tags contained in the response: • "Motor_1" • "Motor_2"

  - Default value: All tags of the system specified by `SystemNames` are queried.

For example:
```
BrowseTags * 100 --filter *Tag0* // browse for tags of all systems
with filter and paging
```

Next request

```
BrowseTags --next
```

Response

OnSuccess
```
NotifyBrowseTags <System>::<Tag name> ... <System>::<Tag name>
```

For example:
```
NotifyBrowseTags HMI_RT_1::InternalTag0 HMI_RT_2::InternalTag01
HMI_RT_2::InternalTag02
```

OnError
```
ErrorBrowseTags <Error description>
```

For example:
```
ErrorBrowseTags Invalid arguments passed to browsing function.
```

BrowseConfiguredAlarms

Description

The "BrowseConfiguredAlarms" command returns the names of the configured alarms of the local HMI system or all HMI systems communicating via Runtime Collaboration.

Information on the general browse behavior of the command can be found in section Behavior of the browse commands (Page 7751).

If the number of hits exceeds the page size of the response, the command consists of an initial request and 1 to N Next requests:

- You define the query in the initial request.
  It provides a response that delivers the first set of hits.
- You can call the remaining hits via Next requests.

Initial request

```
BrowseConfiguredAlarms <System> <Page size> --filter <Filter>
```

- `System`: Optional
  Controls from which system the alarms are read.
  - Value: "*"
    All systems that communicate with each other via runtime collaboration
  - Default value: The local system
- `Page size`: Optional
  Controls how many alarms a response returns.
  Default value: The configured page size is used. See also section ReadConfig (Page 7758).
- `--filter <Filter>`: Optional
  Restricts the command to alarms whose "Name" matches the filter.
  - `--filter`: Command line command
  - `<Filter>`: The filter string
    Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	Filter string: "Motor*:Analog_Alarm" Alarms contained in the response: • "MotorOn:AnalogAlarm" • "MotorOff:AnalogAlarm"
?	Replaces 1 character	Filter string: "Motor_?:Analog_alarm" Alarms contained in the response: • "Motor_1:Analog_alarm" • "Motor_2:Analog_alarm"

  - Default value: All tags of the system specified by `SystemNames` are queried.

For example:
```
BrowseConfiguredAlarms * 100 --filter *Analog* // browse for alarms
of all systems with filter and paging
```

Next request

```
BrowseConfiguredAlarms --next
```

Response

OnSuccess
```
NotifyBBrowseConfiguredAlarms <System>::<Alarm name> ...
<System>::<Alarm name>
```

For example:
```
NotifyBrowseConfiguredAlarms HMI_RT_1::Motor_1:AnalogAlarm_1
HMI_RT_2::Motor_1:AnalogAlarm_1 HMI_RT_2::Motor_2:AnalogAlarm_1
```

OnError
`ErrorBrowseConfiguredAlarms <Error description>`

For example:
```
ErrorBrowseConfiguredAlarms A parameter is not valid or out of
range.
```

BrowseAlarmClasses

Description

The command "`BrowseAlarmClasses`" returns the alarm classes of the local HMI system or all HMI systems communicating via Runtime Collaboration.

Information on the general browse behavior of the command can be found in section Behavior of the browse commands (Page 7751).

Note

Paging is not available for "BrowseAlarmClasses".

Request

`BrowseAlarmClasses <System>`

- `System`: Optional
  Controls from which system the tags are read.
  - Value: "*"
    All systems that communicate with each other via runtime collaboration
  - Default value: The local system

For example:

```
BrowseAlarmClasses  //Browse for alarm classes in local system
```

Response

OnSuccess
`NotifyBrowseAlarmClasses <System>::<Alarm class> ... <System>::<Alarm class>`

For example:
```
NotifyBrowseAlarmClasses HMI_RT_1::Alarm
HMI_RT_1::SystemNotification HMI_RT_1::SystemInformation
HMI_RT_1::SystemAlarm HMI_RT_1::Notification
HMI_RT_1::OperatorInputInformation
```

OnError
`ErrorBrowseAlarmClasses <Error description>`

For example:
```
ErrorBrowseAlarmClasses Invalid arguments passed to browsing
function.
```

See also

ReadConfig (Page 7758)

20.1.4.3 Reference

The following section contains a reference of the properties of tags that you get with the command ReadTagValue.

The commands transfer the property values as string.

Tag properties

"Name" property

Name of the tag

"Value" property

Value of the tag at the moment of the read operation.

"Quality" property

Quality of the read operation of the tag

Possible values:

- "Good"
- "Bad"
- "Uncertain"

"ErrorDescription" property

Description of the error code of the last read or write operation of the tag

20.1.5 Using expert syntax

20.1.5.1 Basics of expert syntax

Characteristics

The expert syntax has the following characteristics:

- Complete JSON commands and replies.
- Commands for individual objects and multiple objects, for example, to write multiple tags in one call.
- Character coding is always UTF-8.
- Cookies are available and mandatory.

Structure of a request

```
{
  "Message": "<Command>",
  "Params":
  {
      "<Object name>":
      [
        "<Param1>",
        "<Param2>"
      ]
  },
  "ClientCookie": "<Cookie name>"
}
```

Structure of a response

OnSuccess

```
{
  "Message": "Notify<Command>",
  "Params":
  {
      "<Object name>":
    [
      {
        "<Value1>"
      },
      {
        "<Value2>"
      }
    ]
  },
  "ClientCookie": "<Cookie name>"
}
```

OnError

```
{
 "Message": "Error<Command>",
```

```
 "ErrorCode": "<Error code>",
 "ErrorDescription": "<Error description>",
 "ClientCookie": "<Cookie name>"
}
```

Overview of the commands of the expert syntax

Command	OnSuccess	OnError	Description
`SubscribeTag <Tag> <ClientCookie>`	`NotifySubscribeTag <Tag name> <Value> <TimeStamp> <Quality> <QualityCode> <hasChanged> <Error code> <Error text> <ClientCookie>`	`ErrorSubscribeTag <Error code> <Error text> <ClientCookie>`	Subscribes one or more tags for monitoring.
`UnsubscribeTag <ClientCookie>`	`NotifyUnsubscribeTag <ClientCookie>`	`ErrorUnsubscribeTag <Error code> <Error text> <ClientCookie>`	Unsubscribes the tag or tags from the cookie from monitoring.
`ReadTag <Tags> <ClientCookie>`	`NotifyReadTag <Tag name> <Value> <TimeStamp> <Quality> <QualityCode> <Errorcode> <Error text> <ClientCookie>`	`ErrorReadTag <Error code> <Error text> <ClientCookie>`	Reads the value of one or more tags of the system.
`WriteTag <Tags, Values> <ClientCookie>`	`NotifyWriteTag <Tag name> <Error code> <Error text> <ClientCookie>`	`ErrorWriteTag <Error code> <Error text> <ClientCookie>`	Writes the specified values to the specified tags.
`SubscribeAlarm <Filter> <Systems> <Language> <ClientCookie>`	`NotifySubscribeAlarm <ClientCookie> <Alarms>`	`ErrorSubscribeAlarm <Error code> <Error text> <ClientCookie>`	Subscribes the alarms defined over the filter, the system and the language ID for monitoring.
`UnsubscribeAlarm <ClientCookie>`	`NotifyUnsubscribeAlarm <ClientCookie>`	`Error UnsubscribeAlarm <Error code> <Error text> <ClientCookie>`	Unsubscribes the alarms from the cookie from monitoring.
`ReadAlarm <Filter> <Systems> <Language> <ClientCookie>`	`NotifyReadAlarm <ClientCookie> <Alarms>`	`ErrorReadAlarm <Error code> <Error text> <ClientCookie>`	Reads the alarms defined by the filter, the system and the LanguageID.

Command	OnSuccess	OnError	Description
`ReadConfig <Parameter> <ClientCookie>`	`NotifyReadConfig <Parameter> <Value> <ClientCookie>`	`ErrorReadConfig <Error code> <Error text> <ClientCookie>`	Reads the settings configured for the general browse behavior.
`WriteConfig <Parameter> <ClientCookie>`	`NotifyWriteConfig <Parameter> <Value> <ClientCookie>`	`ErrorWriteConfig <Error code> <Error text><ClientCookie>`	Sets the settings for the general browse behavior.
`BrowseTags` • Initial request: `BrowseTags <LanguageId> <Filter> <Attribute> <PageSize> <Systems> <ClientCookie>` • Next request: `BrowseTags <Next> <ClientCookie>`	`NotifyBrowseTags <Attribute> <Value> ... <Attribute> <Value> <ClientCookie>`	`ErrorBrowseTags <Error code> <Error text> <ClientCookie>`	Returns "Name", "DisplayName" and "DataType" as well as optional additional attribute values of the tags of the local system or several HMI systems communicating via runtime collaboration.
`BrowseConfiguredAlarms:` • Initial request: `BrowseConfiguredAlarms <LanguageId> <Filter> <Attributes> <PageSize> <SystemNames> <ClientCookie>` • Next request: `BrowseConfiguredAlarms <Next> <ClientCookie>`	`NotifyBrowseConfiguredAlarms <Alarm class> <Alarms> ... <Alarm class> <Alarms> <ClientCookie>`	`ErrorBrowseConfiguredAlarms <Error code> <Error text> <ClientCookie>`	Returns "AlarmClass", "Name" and "Area" as well as optional further attribute values of the configured alarms of an HMI system or several systems communicating via Runtime Collaboration.
`BrowseAlarmClasses <Filter> <Attributes> <Systems>`	`NotifyBrowseAlarmClasses <Alarm class> ... <Alarm class> <ClientCookie>`	`ErrorBrowseAlarmClasses <Error code> <Error text> <ClientCookie>`	Returns the alarm classes of the local system or all systems communicating via Runtime Collaboration.

Errors and error texts

This help only provides a selection of possible error messages. The error texts can also differ from the texts in your projects.

Attributes of the tags and alarms

You can find a description of the attributes of the tags and alarms in the help document Runtime - Open Development Kit (ODK) (Page 7822).

20.1.5.2 Commands

SubscribeTag

Description

The "SubscribeTag" command subscribes one or more tags for monitoring. The following properties are monitored:

- Tag value
- Quality
- Quality code
- Time stamp

"NotifiySubscribeTag" always returns all monitored tags, even if only the value of one monitored tag changes. The change can be a change to the quality code, time stamp or tag value. The order of tags in the response corresponds to the order in the "SubscribeTag" request.

It is permitted to have the same tag monitored by multiple "SubscribeTag" calls.

Request

```
{"Message":"SubscribeTag","Params":{"Tags":
["<Tag>","<Tag>"]},"ClientCookie":"<Cookie>"}
```

- `Tags`: List of the tags to be monitored
- `ClientCookie`: Is used for "UnsubscribeTag" and to assign the notification to its monitoring.

For example:

```
{"Message":"SubscribeTag","Params":{"Tags":
["Tag_0","Tag_1"]},"ClientCookie":"mySubscription1"}
```

Response

OnSuccess

```
{"Message":"NotifySubscribeTag", "Params":{"Tags":[{"Name":"<Tag>",
"Quality":"<Value>", "QualityCode":"<Value>",
"TimeStamp":"<Value>", "Value":"<Tag value>",
"ErrorCode":<Value>, "ErrorDescription":"<Error text>"},
{"Name":"<Tag>", "Quality":"<Value>", "QualityCode":"<Value>",
"TimeStamp":"<Value>", "Value":"<Tag value>", "ErrorCode":<Value>,
"ErrorDescription":"<Error text>"}]}, "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"NotifySubscribeTag", "Params":{"Tags":[{"Name":"Tag_0",
"Quality":"Good", "QualityCode":"192",
"TimeStamp":"2019-01-30T11:25:35Z", "Value":"16", "ErrorCode":0,
"ErrorDescription":""}, {"Name":"Tag_1", "Quality":"Uncertain",
"QualityCode":"76", "TimeStamp":"2019-01-30T11:25:35Z",
```

```
"Value":"1", "ErrorCode":-2147483620, "ErrorDescription":"Tag does
not exist"}]}, "ClientCookie":"mySubscription1"}
```

OnError

```
{"Message":"ErrorSubscribeTag", "ErrorCode":<Value>,
"ErrorDescription":"<Error text>", "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ErrorSubscribeTag", "ErrorCode":-2147483621,
"ErrorDescription":"Subscription could not be created",
"ClientCookie":"mySubscription1"}
```

UnsubscribeTag

Description

The "UnsubscribeTag" command unsubscribes a tag from monitoring that was started with the cookie transferred in the call of "SubscribeTag".

Request

```
{"Message":"UnsubscribeTag","ClientCookie":"<Cookie>"}
```

For example:
```
{"Message":"UnsubscribeTag","ClientCookie":"mySubscription1"}
```

Response

OnSuccess

```
{"Message":"NotifyUnsubscribeTag","ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"NotifyUnsubscribeTag","ClientCookie":"mySubscription1"}
```

OnError

```
{"Message":"ErrorUnsubscribeTag", "ErrorCode":<Value>,
"ErrorDescription":"<Error text>", "ClientCookie":"Cookie"}
```

For example:

```
{"Message":"ErrorUnsubscribeTag", "ErrorCode":-2147483621,
"ErrorDescription":"Subscription could not be closed",
"ClientCookie":"mySubscription1"}
```

ReadTag

Description

The "ReadTag" command reads multiple tags. The tag value, the quality, the quality code and the time stamp are signaled.

The order of tags in the response corresponds to the order in the "ReadTag" request.

Request

```
{"Message":"ReadTag","Params":{"Tags":
["<Tag>","<Tag>"]},"ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ReadTag","Params":{"Tags":
["Tag_0","Tag_1"]},"ClientCookie":"myRequest1"}
```

Response

OnSuccess

```
{"Message":"NotifyReadTag","Params":{"Tags":
[{"Name":"<Tag>","Quality":"<Value>","QualityCode":"<Value>","TimeSt
amp":"<Value>","Value":"<TagValue>","ErrorCode":<Value>,"ErrorDescri
ption":"<ErrorText>"},
{"Name":"<Tag>","Quality":"<Value>","QualityCode":"<Value>","TimeSta
mp":"<Value>","Value":"<TagValue>","ErrorCode":<Value>,"ErrorDescrip
tion":"<ErrorText>"}]},"ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"NotifyReadTag","Params":{"Tags":
[{"Name":"Tag_0","Quality":"Good","QualityCode":"192","TimeStamp":"2
019-01-30T11:25:35Z","Value":"16","ErrorCode":0,"ErrorDescription":"
"},
{"Name":"Tag_1","Quality":"Uncertain","QualityCode":"76","TimeStamp"
:"2019-01-30T11:25:35Z","Value":"1","ErrorCode":-2147483620,"ErrorDe
scription":"Tag does not exist"}]},"ClientCookie":"myRequest1"}
```

OnError

```
{"Message":"ErrorReadTag","ErrorCode":<Value>,"ErrorDescription":"<E
rror text>", "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ErrorReadTag","ErrorCode":-2147483621,"ErrorDescription"
:"Failed to Read", "ClientCookie":"myRequest1"}
```

WriteTag

Description

The "WriteTag" command writes the values of multiple tags.

Request

```
{"Message":"WriteTag","Params":{"Tags":
[{"Name":"<Tag>","Value":"<Tag value>"},
{"Name":"<Tag>","Value":"<Tag value>"}]},"ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"WriteTag","Params":
{"Tags":[{"Name":"Tag_0","Value":"50"},
{"Name":"Tag_1","Value":"40"}]},"ClientCookie":"myRequest2"}
```

Response

OnSuccess

```
{"Message":"NotifyWriteTag","Params":{"Tags":
[{"Name":"<Tag>","ErrorCode":<Value>,"ErrorDescription":"<ErrorText>
"},
{"Name":"<Tag>","ErrorCode":<Value>,"ErrorDescription":"<ErrorText>"
}]},"ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"NotifyWriteTag","Params":{"Tags":
[{"Name":"Tag_0","ErrorCode":0,"ErrorDescription":""},
{"Name":"Tag_1","ErrorCode":-2147483620,"ErrorDescription":"Tag
does not exist"}]},"ClientCookie":"myRequest2"}
```

OnError

```
{"Message":"ErrorWriteTag","ErrorCode":<Value>,"ErrorDescription":"<
Error text>", "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ErrorWriteTag","ErrorCode":-2147483621,"ErrorDescription
":"Failed to Write", "ClientCookie":"myRequest2"}
```

SubscribeAlarm

Description

The "SubscribeAlarm" command subscribes systems for monitoring of changes of active alarms.

The first time "NotifySubscribeAlarm" is called, all active alarms are queried. Afterwards "NotifySubscribeAlarm" is only called if the status of an alarm changes.

Request

```
{"Message":"SubscribeAlarm","Params":{"SystemNames":
["<System>","<System>"],"Filter":"<Filter>","LanguageId":<ID>},"Clie
ntCookie":"<Cookie>"}
```

- `SystemNames`: Optional
  When the list is empty or missing, all known systems are subscribed for monitoring.
- `Filter`: Optional
- `LanguageID`: Optional
- `ClientCookie`:
  Is used for "UnsubscribeAlarm" and to assign the notification to its monitoring.

For example:

```
{"Message":"SubscribeAlarm","Params":{"SystemNames":
["System0","System1"],"Filter":"AlarmClassName !=
'Warning'","LanguageId":1033},"ClientCookie":"CookieForSubscribeAlar
ms123"}
```

Response

OnSuccess

```
{"Message":"NotifySubscribeAlarm","ClientCookie":"<Cookie>","params"
:{"Alarms":[{<Key value pairs for the properties of the first
alarm>},{<Key value pairs for the properties of the second alarm>},
{<...>}]}}
```

For example:

```
{"Message":"NotifySubscribeAlarm","ClientCookie":"CookieForSubscribe
Alarms123","params":{"Alarms":[{"AcknowledgmentTime":"1970-01-01
00:00:00.0000000","AlarmClassName":"Alarm","AlarmClassSymbol":"Alarm
","AlarmText1":"","AlarmText2":"","AlarmText3":"","AlarmText4":"","A
larmText5":"","AlarmText6":"","AlarmText7":"","AlarmText8":"","Alarm
Text9":"","Area":"","BackColor":"4294967295","ChangeReason":"3","Cle
arTime":"1970-01-01
00:00:00.0000000","Connection":"1.0.0.0.0.0","DeadBand":"No
deadband
configured.","Duration":"00:00:01.7431098","EventText":"","Flashing"
:"FALSE","HostName":"md1z5cpc","ID":"0","InfoText":"","InstanceID":"
9","LoopInAlarm":"","ModificationTime":"2019-01-30
11:25:39.9780320","Name":"RUNTIME_1::Tag_2:Alarm2","NotificationReas
on":"1","Origin":"","Priority":"1","RaiseTime":"2019-01-30
11:25:39.9780320","ResetTime":"1970-01-01
00:00:00.0000000","SourceID":"","SourceType":"1","State":"1","StateM
achine":"7","StateText":"R","SuppressionState":"0","SystemSeverity":
"0","Tag":"RUNTIME_1::Tag_2","TextColor":"4278190080","UserName":"",
"Value":"7","ValueLimit":"No limit
configured.","ValueQuality":"192"},
{"AcknowledgmentTime":"1970-01-01
00:00:00.0000000","AlarmClassName":"Alarm","AlarmClassSymbol":"Alarm
","AlarmText1":"","AlarmText2":"","AlarmText3":"","AlarmText4":"","A
larmText5":"","AlarmText6":"","AlarmText7":"","AlarmText8":"","Alarm
Text9":"","Area":"","BackColor":"4294967295","ChangeReason":"3","Cle
arTime":"1970-01-01
00:00:00.0000000","Connection":"1.0.0.0.0.0","DeadBand":"No
deadband
configured.","Duration":"00:00:01.7431098","EventText":"","Flashing"
:"FALSE","HostName":"md1z5cpc","ID":"0","InfoText":"","InstanceID":"
9","LoopInAlarm":"","ModificationTime":"2019-01-30
11:25:39.9780320","Name":"RUNTIME_1::Tag_2:Alarm1","NotificationReas
on":"1","Origin":"","Priority":"1","RaiseTime":"2019-01-30
11:25:39.9780320","ResetTime":"1970-01-01
00:00:00.0000000","SourceID":"","SourceType":"1","State":"1","StateM
achine":"7","StateText":"R","SuppressionState":"0","SystemSeverity":
"0","Tag":"RUNTIME_1::Tag_2","TextColor":"4278190080","UserName":"",
"Value":"7","ValueLimit":"No limit
configured.","ValueQuality":"192", "AlarmGroupID": "1"}]}
```

OnError

```
{"Message":"ErrorSubscribeTag", "ErrorCode":<Value>,
"ErrorDescription":"<Error text>", "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ErrorSubscribeAlarm","ErrorCode":"-2147483621","ErrorDes
cription":"Alarm Subscription failed because of invalid
filter","ClientCookie":"CookieForSubscribeAlarms123"}
```

UnsubscribeAlarm

Description

The "UnsubscribeAlarm" command unsubscribes the alarms from monitoring that was started with the cookie transferred in the call of "SubscribeAlarm".

Request

```
{"Message":"UnsubscribeAlarm","ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"UnsubscribeAlarm","ClientCookie":"CookieForSubscribeAlar
ms123"}
```

Response

OnSuccess

```
{"Message":"NotifyUnsubscribeAlarm","ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"NotifyUnsubscribeAlarm","ClientCookie":"CookieForSubscri
beAlarms123"}
```

OnError

```
{"Message":”ErrorUnsubscribeAlarm","ErrorCode":<Value>,"ErrorDescrip
tion":"<Error text>", "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ErrorUnsubscribeAlarm","ErrorCode":-2147483621,"ErrorDes
cription":"Subscription could not be closed",
"ClientCookie":"CookieForSubscribeAlarms123"}
```

ReadAlarm

Description

The "ReadAlarm" command reads all active alarms.

Request

```
{"Message":"ReadAlarm","Params":{"SystemNames":
["<System>","<System>"],"Filter":"<Value>","LanguageId":<Value>},"Cl
ientCookie":"<Cookie>"}
```

- `SystemNames`: Optional
  When the list is empty or missing, all known systems are subscribed for monitoring.
- `Filter`: Optional
- `LanguageID`: Optional
- `ClientCookie`:
  Is used for "UnsubscribeAlarm" and to assign the notification to its monitoring.

For example:

```
{"Message":"ReadAlarm","Params":{"SystemNames":
["System0","System1"],"Filter":"","LanguageId":1033},"ClientCookie":
"CookieForReadAlarmRequest456"}
```

Response

```
{"Message":"NotifyReadAlarm", "ClientCookie":"<Cookie>","params":
{"Alarms":[{<Key value pairs for properties of the first alarm>},
{<Key value pairs for the properties of the second alarm>},
{<...>}]}}
```

OnSuccess

For example:

```
{"Message":"NotifyReadAlarm",
"ClientCookie":"CookieForReadAlarmRequest456", "params":{"Alarms":
[{"AcknowledgmentTime":"1970-01-01 00:00:00.0000000",
"AlarmClassName":"Alarm",
"AlarmClassSymbol":"Alarm","AlarmText1":"","AlarmText2":"","AlarmTex
t3":"","AlarmText4":"","AlarmText5":"","AlarmText6":"","AlarmText7":
"","AlarmText8":"","AlarmText9":"","Area":"","BackColor":"4294967295
","ChangeReason":"3","ClearTime":"1970-01-01
00:00:00.0000000","Connection":"1.0.0.0.0.0","DeadBand":"No
deadband
configured.","Duration":"00:00:01.7431098","EventText":"","Flashing"
:"FALSE","HostName":"md1z5cpc","ID":"0","InfoText":"","InstanceID":"
9","LoopInAlarm":"","ModificationTime":"2019-01-30
11:25:39.9780320","Name":"RUNTIME_1::Tag_2:Alarm2","NotificationReas
on":"1","Origin":"","Priority":"1","RaiseTime":"2019-01-30
11:25:39.9780320","ResetTime":"1970-01-01
00:00:00.0000000","SourceID":"","SourceType":"1","State":"1","StateM
achine":"7","StateText":"R","SuppressionState":"0","SystemSeverity":
"0","Tag":"RUNTIME_1::Tag_2","TextColor":"4278190080","UserName":"",
"Value":"7","ValueLimit":"No limit
configured.","ValueQuality":"192"},
{"AcknowledgmentTime":"1970-01-01
00:00:00.0000000","AlarmClassName":"Alarm","AlarmClassSymbol":"Alarm
","AlarmText1":"","AlarmText2":"","AlarmText3":"","AlarmText4":"","A
larmText5":"","AlarmText6":"","AlarmText7":"","AlarmText8":"","Alarm
Text9":"","Area":"","BackColor":"4294967295","ChangeReason":"3","Cle
```

```
arTime":"1970-01-01
00:00:00.0000000","Connection":"1.0.0.0.0.0",DeadBand":"No deadband
configured.","Duration":"00:00:01.7431098","EventText":"","Flashing"
:"FALSE","HostName":"md1z5cpc","ID":"0","InfoText":"","InstanceID":"
9","LoopInAlarm":"","ModificationTime":"2019-01-30
11:25:39.9780320","Name":"RUNTIME_1::Tag_2:Alarm1","NotificationReas
on":"1","Origin":"","Priority":"1","RaiseTime":"2019-01-30
11:25:39.9780320","ResetTime":"1970-01-01
00:00:00.0000000","SourceID":"","SourceType":"1","State":"1","StateM
achine":"7","StateText":"R","SuppressionState":"0","SystemSeverity":
"0","Tag":"RUNTIME_1::Tag_2","TextColor":"4278190080","UserName":"",
"Value":"7","ValueLimit":"No limit
configured.","ValueQuality":"192"", "AlarmGroupID": "1"}]}
```

OnError

```
{"Message":"ErrorReadAlarm","ErrorCode":<Value>,
"ErrorDescription":"<Error text>", "ClientCookie":"<Cookie>"}
```

For example:

```
{"Message":"ErrorReadAlarm","ErrorCode":-2147483621,
"ErrorDescription":"Alarm Subscription failed because of invalid
filter", "ClientCookie":"CookieForReadAlarmRequest456"}
```

ReadConfig

Description

The "ReadConfig" command reads the settings configured for the general browse behavior.

Request

```
{"Message": "ReadConfig",  "Params": ['<Parameter>',
'<Parameter>'],  "ClientCookie": '<Cookie>'}
```

- `Params`:
  Value: Comma-separated list of configuration parameters
  Possible parameters:
  - `DefaultPageSize`
    The page size used when a browse request is called without the "PageSize" parameter
  - `BrowseTimeOut`
    Number of seconds after which an inactive browse request is canceled.
- `Cookie`:
  Name of the response cookie

For example:

```
{"Message": "ReadConfig", "Params": ['DefaultPageSize',
'BrowseTimout'], "ClientCookie": "myBrowseAlarmsRequest1"}
```

OnSuccess

```
{"Message":"NotifyReadConfig", "Params": {"<Parameter>":<Value>,
"<Parameter>":<Value>}, "ClientCookie": '<Cookie>'}
```

For example:
```
{"Message":"NotifyReadConfig", "Params": {"DefaultPageSize":500,
"BrowseTimeOut":60}, "ClientCookie": "myBrowseAlarmsRequest1"}
```

OnError
```
{"Message": "ErrorReadConfig",  "ErrorCode": "<Code>",
"ErrorDescription": "<Description>", "ClientCookie": "<Cookie>"}
```

For example:
```
{"Message": "ErrorReadConfig",  "ErrorCode": "-2165322729",
"ErrorDescription": "Invalid arguments passed to browsing
function.", "ClientCookie": "myBrowseAlarmsRequest1"}
```

WriteConfig

Description

The "WriteConfig" command sets configurable settings for the general browse behavior.

Request
```
{"Message": "WriteConfig",  "Params": ["<Parameter>":<Value>,
"<Parameter>":<Value>],  "ClientCookie": '<Cookie>'}
```

- `Params`:
  Value: Comma-separated list of configuration parameters and their values
  Possible parameters:
  - `DefaultPageSize`
    The page size used when a browse request is called without the "PageSize" parameter
    Preset value: 1000
  - `BrowseTimeOut`
    Number of seconds after which an inactive browse request is canceled.
    Preset value: 300 s
- `Cookie`:
  Name of the response cookie

For example:
```
{"Message": "WriteConfig", "Params": ["DefaultPageSize":500,
"BrowseTimout":60], "ClientCookie": "myBrowseAlarmsRequest1"}
```

OnSuccess
```
{"Message":"NotifyWriteConfig", "Params": {"<Parameter>":<Value>,
"<Parameter>":<Value>}, "ClientCookie": '<Cookie>'}
```

For example:
```
{"Message":"NotifyWriteConfig", "Params": {"DefaultPageSize":500,
"BrowseTimeOut":60}, "ClientCookie": "myBrowseAlarmsRequest1"}
```

OnError
```
{"Message": "ErrorWriteConfig",  "ErrorCode": "<Code>",
"ErrorDescription": "<Description>" "ClientCookie": "<Cookie>"}
```

For example:

```
{"Message": "ErrorWriteConfig",  "ErrorCode": "-2165322733",
"ErrorDescription": "A parameter is not valid or out of range.",
"ClientCookie": "myBrowseAlarmsRequest1"}
```

BrowseTags

Description

The "BrowseTags" command returns "Name", "DisplayName" and "DataType" as well as optional further attribute values of the tags of an HMI system or several HMI systems communicating via runtime collaboration.

Information on the general browse behavior of the command can be found in section Behavior of the browse commands (Page 7751).

If the number of hits exceeds the page size of the response, the command consists of an initial request and 1 to N Next requests:

- You define the query in the initial request.
  It provides a response that delivers the first set of hits.
- You can call the remaining hits via Next requests.

Initial request

```
{"Message": "BrowseTags",  "Params": {"LanguageId": <Value>,
"Filter": "<String>", "Attributes": ["<Attribute name>", ...,
"<Attribute name>"], "PageSize": <Value>, "SystemNames":
["<Name>",  ..., "<Name>"]}, "ClientCookie": "<Cookie>"}
```

- `LanguageId`: Optional
  Controls in which language the "DisplayName" of the tags is returned.
  - Value: The language ID
  - Default value: The default language of the system from which the tag originates
- `Filter`: Optional
  Restricts the command to tags whose "Name" matches the filter.
  - Value: Filter string
    Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	Filter string: "Motor*" Tags contained in the response: • "Motor" • "MotorOn" • "MotorOff"
?	Replaces 1 character	Filter string: "Motor_?" Tags contained in the response: • "Motor_1" • "Motor_2"

  - Default value: All tags of the system specified by `SystemNames` are queried.
- `Attributes`: Optional
  Controls which tag attributes returns the response:
  - Value: Comma-separated enumeration of the attribute names
    The response returns "Name", "DisplayName" and "DataType" as well as the specified attributes.
  - Value: "*"
    The response provides all attributes supported by the TIA Portal for tags: Name, DisplayName, AcquisitionMode, Persistent, DataType, Connection, AcquisitionCycle, MaxLength, SubstituteValueUsage, InitialValue, SubstituteValue, InitialMaxValue, InitialMinValue, Address
  - Default value: The response returns "Name", "DisplayName" and "DataType".
- `PageSize`: Optional
  Controls how many tags a response returns.
  Default value: The configured page size is used. See also section ReadConfig (Page 7775).

- `SystemNames`: Optional
  Controls from which system the tags are read.
  - Value: "*"
    All systems that communicate with each other via runtime collaboration
  - Value: Comma-separated list of systems that communicate with each other via runtime collaboration
    FOR EXAMPLE: "HMI_RT_1", "HMI_RT_2"
  - Default value: The local system
- `ClientCookie`:
  Value: Name of the response cookie

For example:
```
{"Message": "BrowseTags",  "Params": {"LanguageId": 1033,
"Filter": "*InternalTag_Bool_1*", "Attributes": ["AcquisitionMode",
"MaxValue"], "PageSize": 50, "SystemNames": ["HMI_RT_1",
"HMI_RT_2"]}, "ClientCookie": "myBrowseTagRequest1"}
```

Next request

```
{"Message": "BrowseTags", "Params": "Next", "ClientCookie":
"<Cookie>"}
```

For example:
```
{"Message": "BrowseTags", "Params": "Next", "ClientCookie":
"myBrowseTagRequest1"}
```

Response

OnSuccess
```
{"ClientCookie": "<Cookie>", "Message": "NotifyBrowseTags",
"Params": {"Tags": [{"<Attribute name":<Value>, "<Attribute
name>":<Value>, "DataType":<Value>, "DisplayName": "<Value>",
"Name":"<Value>"}],}}
```

For example:
```
{"ClientCookie": "<myBrowseTagRequest1>", "Message":
"NotifyBrowseTags", "Params": {"Tags":
[{"AcquisitionMode":0, "MaxValue":1000, "DataType":1,
"DisplayName": "HMI_RT_1::InternalTag_Bool_1",
"Name":"HMI_RT_1::InternalTag_Bool_1"}],}}
```

OnError
```
{"Message": "ErrorBrowseTags", "ErrorCode": "<Code>",
"ErrorDescription": "<Description>", "ClientCookie": "Cookie>"}
```

For example:
```
{"Message": "ErrorBrowseTags", "ErrorCode": "-2165323798",
"ErrorDescription": "Invalid system name.", "ClientCookie":
myBrowseTagRequest1"}
```

BrowseConfiguredAlarms

Description

The "BrowseConfiguredAlarms" command returns "AlarmClass", "Name" and "Area" as well as optional further attribute values of the configured alarms of an HMI system or several systems communicating via Runtime Collaboration.

Information on the general browse behavior of the command can be found in section Behavior of the browse commands (Page 7751).

If the number of hits exceeds the page size of the response, the command consists of an initial request and 1 to N Next requests:

- You define the query in the initial request.
  It provides a response that delivers the first set of hits.
- You can call the remaining hits via Next requests.

Initial request

```
{"Message": "BrowseConfiguredAlarms",  "Params": {"LanguageId":
<Value>, "Filter": "<String>", "Attributes": ["<Attribute
name>", ..., "<Attribute name>"], "PageSize": <Value>,
"SystemNames": ["<Name>", ..., "<Name>"]}, "ClientCookie":
"<Cookie>"}
```

- `LanguageId`: Optional
  Controls in which language the alarm texts are returned.
  - Value: The language ID
  - Default value: The default language of the system from which the alarm originates
- `Filter`: Optional
  Restricts the command to alarms whose "Name" matches the filter.
  - Value: Filter string
    Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	Filter string: "Motor*:Analog_Alarm" Alarms contained in the response: • "MotorOn:AnalogAlarm" • "MotorOff:AnalogAlarm"
?	Replaces 1 character	Filter string: "Motor_?:Analog_alarm" Alarms contained in the response: • "Motor_1:Analog_alarm" • "Motor_2:Analog_alarm"

  - Default value: All alarms of the system specified by `SystemNames` are queried.

- `Attributes`: Optional
  Controls which alarm attributes the response returns:
  - Value: Comma-separated enumeration of the attribute names
    The response returns "AlarmClass", "Name" and "Area" as well as the specified attributes.
  - Value: "*"
    The response returns the following attributes: Name, ID, SourceType, AlarmClassName, Priority, EventText, AlarmText1, AlarmText2, AlarmText3, AlarmText4, AlarmText5, AlarmText6, AlarmText7, AlarmText8, AlarmText9, InfoText, Group, Origin, Area
  - Default: The response returns "AlarmClass", "Name" and "Area".
- `PageSize`: Optional
  Controls how many alarms a response returns.
  Default value: The configured page size is used. See also section ReadConfig (Page 7775).
- `SystemNames`: Optional
  Controls from which system the alarms are read.
  - Value: "*"
    All systems that communicate with each other via runtime collaboration
  - Value: Comma-separated list of systems that communicate with each other via runtime collaboration
    FOR EXAMPLE: "HMI_RT_1", "HMI_RT_2"
  - Default value: The local system
- `ClientCookie`:
  Value: Name of the response cookie

For example:
```
{"Message": "BrowseConfiguredAlarms", "Params": {"LanguageId":
1033, "Filter": "*alarm_*", "Attributes": ["Priority"],
"PageSize": 50, "SystemNames": ["HMI_RT_1"], }, "ClientCookie":
"myBrowseAlarmsRequest1"}
```

Next request

```
{"Message": "BrowseConfiguredAlarms", "Params": "Next",
"ClientCookie": "<Cookie>"}
```

For example:
```
{"Message": "BrowseConfiguredAlarms", "Params": "Next",
"ClientCookie": "myBrowseAlarmsRequest1"}
```

Response

OnSuccess
```
{"ClientCookie": "<Cookie>", "Message":
"NotifyBrowseConfiguredAlarms", "Params":
{"AlarmClasses":[{"AlarmClassName":"<Value>", "Alarms":
[{"AlarmClassName":"<Value>", "Area":<Value>, "Name":<Value>,
"Name":<Value>, "Priority":<Value>}, {"AlarmClassName":"<Value>",
"Area":<Value>, ...}], {"AlarmClassName":"<Value>", "Alarms":
[{"AlarmClassName": "<Value>, ...}]}
```

For example:
```
{"ClientCookie":"myBrowseAlarmsRequest1",
"Message":"NotifyBrowseConfiguredAlarms", "Params":{"AlarmClasses":
[{"Name":"HMI_RT_1::Warning",
"Alarms":[{"AlarmClassName":"HMI_RT_1::Warning",
"Area":"HMI_RT_1::Alarming",
"Name":"HMI_RT_1::Tag6:Analog_alarm_2", "Priority":12},
{"AlarmClassName":"HMI_RT_1::Warning",
"Area":"HMI_RT_1::Alarming",
"Name":"HMI_RT_1::AlarmTag_1:Discrete_alarm_1", "Priority":12}]}]}}
```

OnError
```
{"Message": "ErrorBrowseConfiguredAlarms", "ErrorCode": "<Code>",
"ErrorDescription": "<Description>", "ClientCookie": "Cookie>"}
```

For example:
```
{"Message": "ErrorBrowseAlarms", "ErrorCode": "-2165323798 /
-2165322773", "ErrorDescription": "Invalid system name.", or
"Your browse request has been expired", "ClientCookie":
"myBrowseAlarmsRequest1"}
```

BrowseAlarmClasses

Description

The command "`BrowseAlarmClasses`" returns the alarm classes of the local HMI system or all HMI systems communicating via Runtime Collaboration.

Information on the general browse behavior of the command can be found in section Behavior of the browse commands (Page 7751).

Request
```
{"Message": "BrowseAlarmClasses", "Params": {"Filter": <String>,
"Attributes"["<Attribute name>", ..., "<Attribute name>"],
"SystemNames": ["<Name>", ..., "<Name>"]}, "ClientCookie":
"<Cookie>"}
```

- `Filter`: Optional
  Restricts the command to alarm classes whose "Name" matches the filter.
  - Value: Filter string
    Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	Filter string: "*Alarm" Alarm classes contained in the response: • "Alarm" • "SystemAlarm"
?	Replaces 1 character	Filter string: "Alarm_Prio_? Alarms contained in the response: • "Alarm_Prio_1" • "Alarm_Prio_2"

  - Default value: All alarm classes of the system specified by `SystemNames` are queried.
- `Attributes`: Optional
  Controls which attributes of the alarm class return the response:
  - Value: Comma-separated enumeration of attributes
    The response returns "Name" and "StateMachine" as well as the specified attributes.
  - Value: "*"
    The response returns the following attributes: Name, StateMachine, ID, Priority, NormalStateTextColor, NormalStateBackColor, RaisedStateTextColor, RaisedStateBackColor, RaisedStateFlashing, AcknowledgedStateTextColor, AcknowledgedStateBackColor, AcknowledgedStateFlashing, ClearedStateTextColor, ClearedStateBackColor, ClearedStateFlashing, AcknowledgedClearedStateTextColor, AcknowledgedClearedStateBackColor, AcknowledgedClearedStateFlashing
  - Default: The response returns "Name" and "StateMachine".
- `System`: Optional
  Controls from which system the tags are read.
  - Value: "*"
    All systems that communicate with each other via runtime collaboration
  - Value: Comma-separated list of systems that communicate with each other via runtime collaboration
    FOR EXAMPLE: "HMI_RT_1", "HMI_RT_2"
  - Default value: The local system

For example:

```
{"Message": "BrowseAlarmClasses", "Params": {"Filter":
"*Alarm*", "SystemNames": ["HMI_RT_1"]}, "ClientCookie":
"myBrowseAlarmClassRequest1"} //Browse for alarm classes in
specified system with filter
```

Response

OnSuccess

```
NotifyBrowseAlarmClasses <System>::<Alarm class> ...
<System>::<Alarm class>
```

For example:
```
NotifyBrowseAlarmClasses HMI_RT_1::Alarm
HMI_RT_1::SystemNotification HMI_RT_1::SystemInformation
HMI_RT_1::SystemAlarm HMI_RT_1::Notification
HMI_RT_1::OperatorInputInformation
```

OnError
```
ErrorBrowseAlarmClasses <Error description>
```

For example:
```
ErrorBrowseAlarmClasses Invalid arguments passed to browsing
function.
```

20.1.5.3 Reference

The following section contains a reference of the properties of alarms and tags that you get with the commands ReadAlarm and ReadTag.

The commands transfer the property values as string.

Tag properties

"Name" property

Name of the tag

"Value" property

Value of the tag at the moment of the read operation.

"Quality" property

Quality of the read operation of the tag

Possible values:

- "Good"
- "Bad"
- "Uncertain"

"QualityCode" property

Quality code of the read operation of the tag

"TimeStamp" property

Time stamp of the last successful read operation of the tag

"Error" property

Error code of the last read or write operation of the tag

"ErrorDescription" property

Description of the error code of the last read or write operation of the tag

Alarm properties

"InstanceID" property

InstanceID for an alarm with multiple instances

"SourceID" property

Source at which the alarm was triggered.

"Name" property

Name of the alarm

"AlarmClassName" property

Name of the alarm class

"AlarmClassSymbol" property

Symbol of the alarm class

"AlarmParameterValues" property

Parameter values of the alarm

"AlarmText1" ... "AlarmText9" properties

Additional texts 1-9 of the alarm

"ChangeReason" property

Trigger event of the modification of the alarm state

"Connection" property

Connection via which the alarm was triggered.

"State" property

Current alarm state

The property can contain the following values:

- "0": Normal
- "1": Raised
- "2": RaisedCleared
- "5": RaisedAcknowledged
- "6": RaisedAcknowledgedCleared
- "7": RaisedClearedAcknowledged
- "8": Removed

"StateText" property

Current alarm state as text, e.g. "active" or "inactive"

"EventText" property

Text that describes the alarm event.

"InfoText" property

Text that describes an operator instruction for the alarm.

"TextColor" property

Number with the text color of the alarm state

"BackColor" property

Number with the background color of the alarm state

"Flashing" property

Indicates whether the alarm flashes.

Values: "TRUE" or "FALSE"

"ModificationTime" property

Time of last modification to the alarm state

"RaiseTime" property

Trigger time of the alarm

"AcknowledgementTime" property

Time of alarm acknowledgment

"ClearTime" property

Time of alarm reset

"ResetTime" property

Time of alarm reset

"SuppressionState" property

Status of alarm visibility

"SystemSeverity" property

Severity of the system error

"Priority" property

Relevance for display and sorting of the alarm

"Origin" property

Origin for display and sorting of the alarm

"Area" property

Origin for display and sorting of the alarm

"Value" property

Current process value of the alarm

"ValueQuality" property

Quality of the process value of the alarm

"ValueLimit" property

Limit of the process value of the alarm

"UserName" property

User name of the operator control alarm

"HostName" property

Name of the host that triggered the alarm.

"ID" property

ID of the alarm that is also used in the display.

"AlarmGroupID" property

ID of the alarm group to which alarm belongs.

"SourceType" property

Source from which the alarm was generated, e.g. tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

The enumeration "HmiAlarmSourceType" can contain the following values:

- `Undefined` (0)
- `Tag` (1)
- `Controller` (2)
- `System` (3)
- `Alarm` (4)

"DeadBand" property

Range of the triggering tag, in which no alarms are generated.

```
object DeadBand { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm control to its origin.

```
string LoopInAlarm { get; }
```

"NotificationReason" property

Reason for the notification

The property can contain the following values:

- `"0": Unknown`
- `"1": Add`
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- `"2": Modify`
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `"3": Remove`
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarms will not result in notifications until the alarm again meets the filter criteria. In this case, "NotificationReason" is set to `Add`.

Note

Removing an alarm from business logic

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- State-based monitoring: The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.

Example:

A customer application begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the customer application of the "NotificationReason" as follows:

NotificationReason	Description
Add	• The "State" property is 1. The alarm is active.
Modify	• The "State" property has not changed. • Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, e.g. alarm is inactive.

"Duration" property

Returns the time interval in nanoseconds between triggering of the alarm and its previous status change.

20.1.5.4 Syntax of the alarm filter

With an AlarmSubscription, a filter can be transferred so that not all active alarms of the alarm system are notified, but only those which match the filter. The filter syntax is based on SQL syntax. However, only the WHERE instruction is relevant. The keyword "WHERE" must be omitted.

Operators

The following operators can be used in the filter string of the alarm filter:

Operator	Description	Example
=	equal to	`Name = 'Recipe246'`
<>	not equal	`Value <> 0.0`
>	greater than	`Value > 25.0`
<	less than	`Value < 75.0`
>=	greater than or equal to	`Value >= 25.0`
<=	less than or equal to	`Value <= 75.0`
`OR, ||`	logical OR	`State = 1 OR State = 3`
`AND, &&`	logical AND	`Value >= 25.0 AND Value <= 75.0`
`BETWEEN`	within a range	`Value BETWEEN 25.0 AND 75.0`
`NOT BETWEEN`	outside a range	`Value NOT BETWEEN 25.0 AND 75.0`
`LIKE string`	corresponds to the string *string*	`Name LIKE 'Motor*'`
`NOT LIKE string`	does not correspond to the string *string*	`Name NOT LIKE 'Valve*'`
`IN (v1, v2, …)`	corresponds to one or more values	`State IN (1, 4, 7)`
`NOT IN (v1, v2, …)`	does not correspond to one or more values	`State NOT IN (0, 2, 3, 5, 6)`
`(…)`	brackets expressions	`Value <= 75.0 AND (State = 1 OR State = 3)`

Precedence of the operators:

Rank	Operators
1	• Relational operators: =, <>, >, <, >=, <= • LIKE • IN • BETWEEN
2	NOT
3	AND, &&
4	OR, ||

Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	`Name LIKE 'Motor*'` `Reference = <1.*.15>1`
?	Replaces 1 character	`Name = 'Recipe?'`

20.2 Programming Custom Web Controls

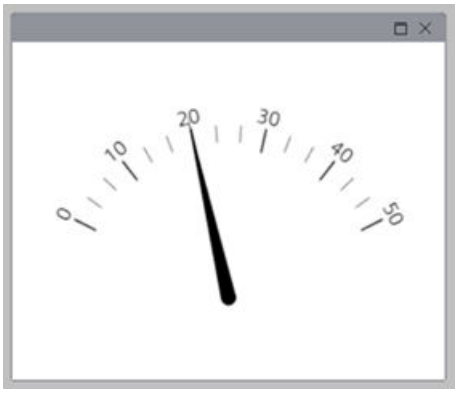
20.2.1 Custom web controls

Custom Web Controls represent an independent web page with interface to Runtime. Custom Web Controls offer you the option of adding your own elements to the visualization elements provided. Custom Web Controls thus extend usability and functionality to achieve an optimal visualization result.

Custom web controls are run on the web client and hosted in Runtime. A Custom Web Control can be displayed as an independent Web page in any browser and on any mobile device.

Note

Observe the performance limit of Unified Comfort Panels. We do not recommend using, for example, Custom Web Controls with 3D representations for Unified Comfort Panels.



Requirements for a web-based graphic interface

To use a Custom Web Control in WinCC, the Control must be anchored in a container. The container is provided on the user side by the Custom Web Control framework and contains components of the graphical user interface (GUI).

The following requirements apply when a web-based GUI component if it is to be provided in a Custom Web Control container:

- The component must be HTML5-based and interpretable by current browsers.
- The component must be executable exclusively on the client side.
- The component must work without interaction with client-side components outside the container.
- The component must comply with the principle of a Single Page Application (SPA) and fit on a web page. All code (HTML, JavaScript and CSS) must be received when the page is called or dynamically added during user actions. The web page may not reload at any time.
- The component must not know in which environment it is deployed. The component must be executable independent of the environment.
- All data exchange must take place through communication between client and server.

Application example

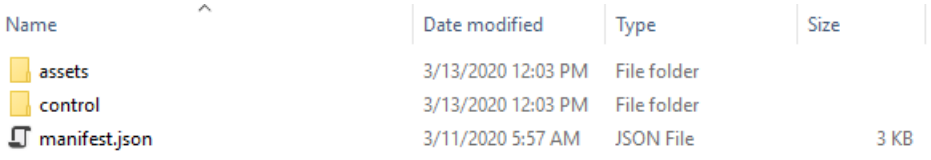
Based on an application example, you can learn more about the structure and development of a Custom Web Control: SIOS entry (https://support.industry.siemens.com/cs/ww/en/view/109779176).

20.2.2 General structure and folder structure

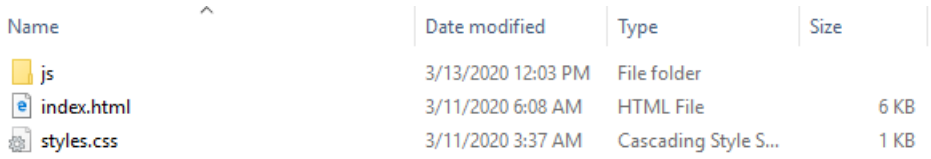
A ready-to-use Custom Web Control must be available as a "*.zip" file that contains all graphics and code files used. The structure is divided into two folders, "assets" and "control", and a "*.json" file (manifest.json).

The "assets" folder contains a logo that is displayed in the TIA Portal. The "control" folder contains "*.html", "*.js" and "*.css" files, as well as used graphics and icons that the Control needs for the display.

A Custom Web Control has the following folder structure:

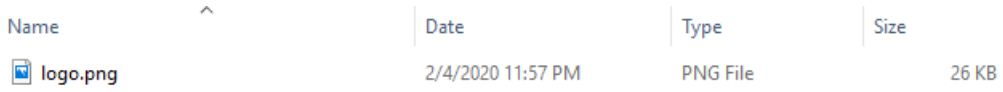


Contents of the "control" folder:



Name	Date modified	Type	Size
js	3/13/2020 12:03 PM	File folder	
index.html	3/11/2020 6:08 AM	HTML File	6 KB
styles.css	3/11/2020 3:37 AM	Cascading Style S...	1 KB

Contents of the "assets" folder:



Name	Date	Type	Size
logo.png	2/4/2020 11:57 PM	PNG File	26 KB

20.2.3 Contract-based interaction and the manifest file

20.2.3.1 Basics for the manifest

To enable the Unified Runtime server to communicate with the provided control, the control must reveal methods, events and properties to the Unified Runtime server.

The sum of the information that the control releases with it is called the "contract". For the custom web control container, this information is contained in a "*.json" file (manifest.json). The manifest file contains multiple sectors that each reveal elements to the container.

Tips for an efficient procedure

To verify that you have created a valid JSON file, use the notes in Visual Studio Code or copy the content of the file to an online validation tool, such as "https://jsonlint.com".

20.2.3.2 Manifest structure

Each manifest has two root elements:

- "mver": Specifies the manifest version.
- "control": Specifies the manifest type.

The "control" element provides the following sectors.

- "identity" sector
- "environment" sector
- "metadata" sector

- "contracts" sector
  The "contracts" sector contains the following:
  - Methods
  - Events
  - Properties
- "types" sector

"identity" sector

The "identity" sector contains identity information.

The following information of the data type "String" is required:

- "name": Defines the name of the Custom Web Control.
- "version": Defines the version of the Custom Web Control.
- "displayname": Defines the display name of the Custom Web Control.

  Note

  Special characters must not be used, e.g. #,$,*,%,.,/,;,?;[,],~,'".

- "icon" (optional)
  Includes the path for the logo that is displayed in the "Toolbox > My Controls" task card in the TIA Portal.
  The path can be specified as follows:
  - URL starting with "http://" or "https://"
  - Relative path to the storage location of the manifest starting with "./"
  - Data URL that contains a Base64 encoded image starting with "data:"

  The referenced image must be between 120x120 pixels and 320x320 pixels in size. The following image formats are supported:
  - JPG and JPEG
  - PNG
  - ICO
  - TIFF
  - BMP

  Use a square image to avoid distortions.
  If "icon" is not specified, only the display name ("displayname") is shown under "My Controls".

- "type": Each Custom Web Control can be referenced via the identity type and therefore requires a pre-defined structure. Types must follow the 8-4-4-4-12 pattern of a 128-bit integer (GUID).

  Note

  GUID generation

  For example, a GUID can be created as follows:
  - In Visual Studio under "Tools > Create GUID"
  - Under https://www.guidgenerator.com/ (https://www.guidgenerator.com/)

- "start" (optional): The start directory must be specified to set the starting point of the Custom Web Control for the browser.
  If this value is not defined, "./control/index.html" is used.

Example: "identity"

```
"identity": {
   "name": "GaugeMeter",
   "version": "1.0",
   "displayname": "GaugeMeter",
   "icon": "./assets/logo.ico",
   "type": "guid://551BF148-2F0D-4293-99C2-C9C3A1A6A073",
   "start": "./control/index.html"
}
```

"environment" sector

The optional "environment" sector provides information on the environment that is integrated into the Custom Web Control. If the sector does not exist, then no requirements and dependencies exist.

Requirements are specified under the element "prerequesites". "renderingspace" can be used below "prerequesites". Restrictions of the display of the Custom Web Control can be specified under "renderingspace".

The following restrictions are permitted:

Restriction	Data type	Description
minwidth	integer	Specifies the minimum width that is necessary to display the Custom Web Control.
maxwidth	integer	Specifies the maximum width that is permitted to display the Custom Web Control.
defaultwidth	integer	Specifies the default width of the Custom Web Control. The value must be between "minwidth" and "maxwidth".
minheight	integer	Specifies the minimum height that is necessary to display the Custom Web Control.

Restriction	Data type	Description
maxheight	integer	Specifies the maximum height that is permitted to display the Custom Web Control.
defaultheight	integer	Specifies the default height of the Custom Web Control. The value must be between "minheight" and "maxheight".
unit	string	Specifies the unit of the display restrictions. The following values are permitted: • "px" • "cm" • "mm" • "in" • "pt" If "unit" is not specified, "px" is used.

Extensions can be specified under "extensions".

Each extension has the following optional fields:

- "mandatory": Specifies whether the Custom Web Control can be used without the extension. Regardless of the value in this field, extensions of a Custom Web Control are always necessary.
- "version": Specifies a compatible version or versions.

You can find available extensions at Extensions (Page 7804).

Example: "environment"

```
"environment": {
  "prerequisites": {
    "renderingspace": {
      "defaultwidth": 450,
      "defaultheight": 300,
      "unit": "px"
    }
  }
  "extensions": {
    "HMI": {
      "mandatory": true,
      "version": "~1.0.0"
    }
  }
}
```

"metadata" sector

The following optional information can be stored, for example, as metadata:

- Author
- Keywords

- Description of the Custom Web Control
- Homepage

Metadata are not relevant for the execution of the Custom Web Control. User-defined metadata can be supplemented.

Example: "metadata"

```
"metadata":{
  "author": "Siemens",
  "keywords": [
    "Gauge",
    "GaugeMeter"
  ]
  "description": "Display tag value with a gauge."
"homepage": "https://www.siemens.com"
"company": "Siemens AG"
}
```

"contracts" sector

The "contracts" sector contains methods, events and properties as interface for use in the TIA Portal.

The Custom Web Control has access to methods, events and properties and therefore receives tag changes from the PLC, for example.

Data types can be used or referenced in this sector.

Note

A data type can be assigned with the keyword "type" below an element or it can be referenced with "$ref". The two keywords cannot be used at the same time.

You can find additional information at Data types and references in the manifest (Page 7800).

Note the following restrictions when naming methods, events, properties, arguments and parameters:

- Only alphanumeric characters of the ASCII character set and "_" are permitted.
- The names must not start with a number.
- The entry is case-sensitive.
- Special characters are not permitted.

Methods

"methods" contain a list of methods used by the Custom Web Control.

Methods of a Custom Web Control can be used in Unified Scripting to transfer information from the Unified server to the custom web control in the client. Methods are always executed asynchronously.

For example, a method to flash the passed zone can be called as follows:

```
Screen.Items('GaugeMeter_1').BlinkZone(2)
```

The following optional elements can be assigned to a method:

- "return": Specifies or references a data type.
  An additional "promise" field of the type "Boolean" returns whether the method is actually running asynchronously and is not fulfilled in a defined time period. The default value is "false". If the value is "true", the type specified with "return" is valid for the fulfilled value of the Promise object.
  You can react as follows to the Promise object:
  `"const result = await Screen.Items('GaugeMeter_1').BlinkZone(2) HMIRuntime.Trace(result);"`
- "parameters": Each parameter specifies or references a data type.
- "description": Specifies the description of the method.

Example: "methods"

```
"methods": {
  "BlinkZone": {
    "parameters": {
      "zoneIndex": {
        "type": "number"
      }
    },
    "description": "Let the given zone blink."
  }
}
```

Events

"events" contain a list of events used by the Custom Web Control.

Events are triggered by the Custom Web Control itself at any time. Events can be used in Unified Scripting to transfer information from the client to the server. They can be found in the engineering system under "Properties > Events".

An event can be assigned the following optional elements:

- "arguments": Contains arguments. Each argument specifies or references a data type.
- "description": Specifies the description of the event.

Example: "events"

```
"events": {
  "ZoneChanged": {
    "arguments": {
      "zoneIndex": {
        "type": "number"
      }
    },
    "description": "Whenever the zone is changed, this event is raised and
gives you the new zone index."
  }
}
```

Properties

"properties" contain a list of properties used by the Custom Web Control.

Properties can be found in the engineering system under "Properties > Properties > Interfaces".

The following elements can be assigned to a property:

- "type" or "$ref": A data type must be specified or referenced.

  Note

  Only individual elements can be connected

  When using complex data types, such as an array or user data types, only the elements of the bottom level can be linked to a property.

- "default" (optional): Specifies the default value of the property.
- "description" (optional): Specifies the description of the property.

Example: "properties"

```
"properties": {
  "GaugeValue": {
    "type": "number",
    "default": 20
    "description": "This property represents the value of the gauge."
  },
  "MinValue": {
    "type": "number",
    "default": 0
   "description": "This property represents the minimum value of the gauge."
  },
  "MaxValue": {
    "type": "number",
    "default": 50
   "description": "This property represents the maximum value of the gauge."
  },
}
```

"types" sector

The "types" sector contains local definitions for user-defined data types, objects and arrays.

A data type that is defined in this sector can only be referenced within this manifest. External data types from a JSON Schema can be referenced under "contract" or "types".

Example: "types"

```
"types": {
  "Color": {
    "$id": "http://tia.siemens.com/wincc-unified/types/s/color",
    "type": "number"
  },
  "AlignmentPart": {
    "type": "object",
    "properties": {
      "Vertical": {
        "$ref":"#/control/types/VerticalAlignment"
      }
    }
  },
  "VerticalAlignment": {
    "type": "string",
    "enum": [
      "Top",
      "Center",
      "Bottom"
    ],
    "default": "Center"
  }
}
```

See also

Creating the ZIP file (Page 7819)

20.2.3.3 Data types and references in the manifest

Data types

Data types in the "contracts" sector of the manifest can be specified or referenced as follows:

- As basic data type
  The following basic data types can be used without reference:
  - Boolean: Can accept "true" or "false".
  - Number: Any representable number, for example:
    Integer: 1; -3
    Float: 5.21
    Numbers in exponential format: 2.99792458e8
  - String: Any text
  - Null: Displays the zero object and is used as return type for methods without return value.
- As local reference to the "types" sector of the manifest
  Arrays, objects or user-defined data types can be defined in the "types" sector.
- As external reference to a JSON Schema that is offered by Siemens, for example:
  - "$id": "http://tia.siemens.com/wincc-unified/types/s/color"
  - "$id": "http://tia.siemens.com/wincc-unified/types/c/font"

User-defined data types are, for example, structured data types or basic data types with restrictions.

The following type restrictions are permitted in the manifest:

Type restriction	Description
enum	Specifies the permitted String values of an enumeration.
items	Specifies the permitted data type of an array.
minItems	Specifies the minimum number of elements in an array.
maxItems	Specifies the maximum number of elements in an array.
minimum	Specifies the minimum value of a number of the data type "number" or "integer".
maximum	Specifies the maximum value of a number of the data type "number" or "integer".
pattern	Specifies the permitted pattern as regular expression that defines the content of a string.
minLength	Specifies the minimum length of a string.
maxLength	Specifies the maximum length of a string.
required	Specifies a list of mandatory properties within a structure.

Identification of a data type

When you define a data type in the "types" sector, you can specify a Uniform Resource Identifier (URI) or a fragment in the optional "$id" field. This information uniquely identifies the data type across multiple manifests. Within the manifest in which a data type is defined, this data type can also be referenced by using its ID.

Note

Absolute URIs are preferred over fragments.

Referencing a data type

Data types in the "contracts" or "types" sector can be referenced with "$ref".

You have the following options for referencing data types:

- As reference to the "types" sector without using the "$id"
  Example: `"BackColor":{"$ref": "#/control/types/color"}`
- As reference to the "types" sector using the "$id"
  Example: `"BackColor":{"$ref": "http://tia.siemens.com/wincc-unified/types/s/color"}`

20.2.4 Interaction between control and container via the API

A single API object ("WebCC" object) is used to enable communication between the Custom Web Control and the Unified Runtime server.

The following requirements apply to this API object:

1. All functionalities that the Control needs for independent executability must be available on the client side.
2. The API object must be created and extended with the specific functionalities that the control provides through the manifest file.

The API object

The API object represents the interface through which the methods, events and properties of the Control are called or received from the framework.

For the initialization of the Custom Web Control, the properties, methods and events of the manifest file must be declared.

You can link the properties, methods and events to your code.

Note

Declaration

The name must match the name of the manifest file.

Note the following restrictions:

- Only alphanumeric characters of the ASCII character set and "_" are permitted.
- The names must not start with a number.
- The entry is case-sensitive.
- Special characters are not permitted.

Example: API object

```
{
  //Methods
  methods: {
    BlinkZone: function(zoneIndex){
      //code
    }
  },
  //Events
  events: ['ZoneChanged', 'Event2'],
  //Properties
  properties:{
    GaugeValue: " ",
    Property2: " "
  }
}
```

Integrating the "WebCC" object

Integrating the API object makes the corresponding namespaces available to the Custom Web Control.

As a requirement, a JavaScript file (webcc.min.js) must be integrated in "index.html", which performs a handshake between the control and the container.

CAUTION

Do not change "webcc.min.js" file

The "webcc.min.js" file is used to establish the connection and must not be changed.

The "index.html" file is the entry point of your web page.

Note

Download "webcc.min.js"

You can find the file "webcc.min.js" in the application example: SIOS entry (https://support.industry.siemens.com/cs/ww/en/view/109779176)

The "webcc.min.js" file contains the connection data to WinCC Unified.

After the connection setup, you can access the data that is defined in the "manifest.json" file from any position in your application.

Example: Integrating the "WebCC" object

```
<!doctype html>
<head>
  <script>…</script>
  <!-- Web Custom Control Facade -->
  <script type= text/JavaScript src='webcc.min.js'>
</head>
```

Initialization of the "WebCC" object

The API object must have been successfully initialized before the Control can be created. To check this, the tag "result" of the parameter "function(result)" is queried. It is used as an indicator and must supply "true" to continue. As an additional parameter, extensions can be called during initialization.

Example: Initialization of the "WebCC" object

```
WebCC.start( function( result ) {
      if ( result ) {
        //startup succeeded
        //add subscriptions
      } else {
        //startup failed
      }
    },
controlInit.ControlApi,
  ['HMI'] };
```

See also

Extensions (Page 7804)

20.2.5 Extensions

20.2.5.1 Basics of extensions

With extensions you can use additional functions in Custom Web Controls.

The following extensions can be used:

- HMI extension:
  - Obejct "WebCC.Extensions.HMI.Properties" provides access to all properties of the container that contains the Custom Web Control.
  - Object "WebCC.Extensions.HMI.Style" allows access to the active style
  - The "DatePrecise", "Big", and "Variant" prototypes as extensions to the integrated JavaScript objects.
- Formatting extension: The "WebCC.Extensions.Formatting.Output" object enables the formatting of text.
- Dialog extension: Allows access to a dialog window.

When using extensions, follow these steps:

1. You integrate the extensions used in the manifest in the "environment" sector.
   This permits the environment to check whether the respective extension exists.
2. You call the extensions used during the initialization of the "WebCC" object.
3. Call the extension as property of the "WebCC" object at any point in your Custom Web Control.

Integration in the manifest

Integrate extensions in the "environment" sector of the manifest as follows:

```
"environment": {
  "extensions": {
    "HMI": {
      "mandatory": true,
      "version": "~1.0.0"
    }
    "Formatting": {
      "mandatory": true,
      "version": "~1.0.0"
    }
    "Dialogues": {
      "mandatory": true,
      "version": "~1.0.0"
    }
  }
}
```

Initialization of the "WebCC" object

```
WebCC.start( function( result ) {
      if ( result ) {
        //startup succeeded
        //add subscriptions
      } else {
        //startup failed
      }
    },
controlInit.ControlApi,
  ['HMI', 'Formatting', 'Dialogues'] };
```

See also

Dialog extension (Page 7810)

Formatting extension (Page 7809)

HMI extension (Page 7805)

20.2.5.2 HMI extension

"Properties" object

The "WebCC.Extensions.HMI.Properties" object provides access to all properties of the container that contains the Custom Web Control. The properties are placed after the "Properties" object.

You can find all available properties in the WinCC Unified object model under AUTOHOTSPOT.

The extension enables the "WebCC.onPropertyChanged.subscribe" event that can be used to register for changes of properties.

"Style" object

The "WebCC.Extensions.HMI.Style" object has the "Name" property that contains the name of the active style.

The "WebCC.Extensions.HMI.Style.onChanged" event is initiated when the style is changed in runtime. The event contains the name of the style that is being activated.

Example

```
WebCC.Extensions.HMI.Style.onChanged.subscribe( function( currentStyle) {
console.log(currentStyle);
});
```

Data types

Introduction

Prototypes are provided as extension of the installed JavaScript objects; they are used in the HMI environment.

The prototypes are available in the global namespace.

"DatePrecise" prototype

The JavaScript object "Date" provides millisecond accuracy, while Unified Runtime works with nanosecond accuracy.

The "DatePrecise" object is used to work with nanosecond accuracy. The "DatePrecise" object does not contain any information about time zones and uses coordinated universal time (UTC) for calculations.

Constructors and methods

Constructor	Description
`DatePrecise()`	Creates a "DatePrecise" object for the current date and time.
`DatePrecise(year, month[, day[, hours[, minutes[, seconds[, milliseconds[, microseconds[, nanoseconds]]]]]]])`	Creates a "DatePrecise" object. This constructor corresponds to the JavaScript "Date" constructor, but enables additional parameters for microseconds and nanoseconds. The parameters "year" and "month" are mandatory.
`DatePrecise(DomHighResTimeStamp)`	Creates a "DatePrecise" object that accepts the "DomHighResTimeStamp" object of a browser with a millisecond value since January 1, 1970, and an accuracy in the microsecond range.
`DatePrecise([seconds, nanoseconds])`	Creates a "DatePrecise" object that accepts the second value since January 1, 1970, and an additional nanosecond offset.
`DatePrecise(date)`	Creates a "DatePrecise" object from a passed JavaScript "Date" object. The nanoseconds are "0".
`DatePrecise(precise)`	Creates a "DatePrecise" object from a passed "DatePrecise" object.

Method	Description
`getMicroseconds()`	Returns the number of microseconds (from 0 to 999).
`getNanoseconds()`	Returns the number of nanoseconds (from 0 to 999).
`getTime()`	Returns the number of milliseconds since January 1, 1970. This method corresponds to the method of the JavaScript "Date" object.

Method	Description
`getHrTime()`	Returns a precise date as an array with two numerical values. The first value represents the seconds elapsed since January 1, 1970. The second value is the offset in nanoseconds.
`setMicroseconds(microseconds)`	Sets the microseconds (from 0 to 999).
`setNanoseconds(nanoseconds)`	Sets the nanoseconds (from 0 to 999).
`setTime(DomHighResTimeStamp)`	Determines the date over a specific number of milliseconds, starting from January 1, 1970. This method corresponds to the method of the JavaScript "Date" object.
`setHrTime([seconds, nanoseconds])`	Specifies a precise date as array with two numerical values. The first value represents the seconds elapsed since January 1, 1970. The second value is the offset in nanoseconds.
`toDate()`	Returns a "Date" object. The nanosecond accuracy is lost.
`valueOf()`	Returns a "DomHighResTimeStamp" object. The nanosecond accuracy is lost. The object is compatible with the JavaScript "Date" object.

Examples

```
var ms = window.DatePrecise([1593862222, 545410000]).getTime();
var date = window.DatePrecise(Date.UTC(1960, 11, 24, 18, 4, 5, 10));
var jsdate = date.toDate();
```

"Big" prototype

The "Big" prototype enables processing of large numbers with any desired precision. You can find the library and application examples at "Big.js".

Constructors and methods

Note

The "n" parameter can take on the following data types:

- JavaScript data type "Number"
- "Big" data type
- "String" data type that contains a numerical value

Constructor	Description
`Big(n)`	Creates a number of the data type "Big" with the value "n".

Method	Description
`abs()`	Returns the absolute value of the "Big" number.
`cmp(n)`	Compares two "Big" numbers. The following return values occur: • If the "Big" number is greater than "n", 1 is returned. • If the "Big" numbers are the same, 0 is returned. • If the "Big" number is less than "n", -1 is returned.
`div(n)`	Returns the value of the "Big" number divided by "n".
`eq(n)`	Returns a Boolean value that indicates whether the "Big" number and "n" are the same.
`gt(n)`	Returns a Boolean value that indicates whether the "Big" number is greater than "n".
`gte(n)`	Returns a Boolean value that indicates whether the "Big" number is greater than or equal to "n".
`lt(n)`	Returns a Boolean value that indicates whether the "Big" number is less than "n".
`lte(n)`	Returns a Boolean value that indicates whether the "Big" number is less than or equal to "n".
`minus(n)`	Returns the value of the "Big" number minus "n".
`mod(n)`	Returns the value of the "Big" number using modulo of value "n".
`plus(n)`	Returns the value of the "Big" number plus "n".
`pow(exp)`	Returns the value of the "Big" number to the power of "exp". The value for "exp" must be an integer between -1e+6 and +1e+6.
`round([dp [, rm]])`	Returns the rounded value of the "Big" number to a maximum of "dp". The value for "dp" must be an integer between -1e+6 and +1e+6. The default value for the parameter "dp" is 20. The "rm" parameter supports the following modes: • 0: Rounded to zero. • 1: Rounded to the nearest neighbor. If the distance is equal, the figures are rounded up. • 2: Rounded to the nearest neighbor. If the distance is the same, it is rounded to the straight neighbor. • 3: Rounded up.
`sqrt()`	Returns the square root of the "Big" number.
`times(n)`	Returns the value of the "Big" number multiplied by "n".
`toExponential([dp])`	Returns a string that represents the value of the "Big" number type in exponential notation. The "dp" parameter defines the number of decimal places displayed. The value for "dp" must be an integer between 0 and 1e+6. The default value for the parameter "dp" is 20.
`toFixed([dp])`	Returns a string that represents the value of the "Big" number type in standard notation. The "dp" parameter defines the number of decimal places displayed. The value for "dp" must be an integer between 0 and 1e+6. The default value for the parameter "dp" is 20.

Method	Description
`toPrecise(sd)`	Returns a string that represents the number of significant digits of the "Big" number. The "sd" parameter defines the number of decimal places displayed. The value for "sd" must be between 1 and 1e+6. The default value for the parameter "sd" is 20. If the "Big" number has more than the number of significant digits defined by "sd", the return value is rounded to the number of significant digits defined by 'sd'. The rounding mode is 1.
`toString()`	Returns a string that represents the value of the "Big" number. Under the following conditions, the value is displayed in exponential notation: • The positive exponent is greater than or equal to 21. • The negative exponent is less than or equal to -7.

"Variant" prototype

Basic data types can be mapped with the "Variant" prototype.

Constructors and methods

Constructor	Description
`Variant(value, type)`	Creates a "Variant" object using any value and data type.

Methods	Description
`typeOf()`	Returns a numerical value that identifies the elementary data type.
`valueOf()`	Returns the value that was converted into the corresponding data types "Number", "Boolean", "String", "DataPrecise" or "Big". "Time" and "DateTime" are returned as values of the type "DatePrecise". Numerical values that cannot be mapped without loss of accuracy are returned as values of the type "Big".

Examples

```
var variant = window.Variant(47111, 0x5);
var big = variant.valueOf();
```

20.2.5.3 Formatting extension

The "WebCC.Extensions.Formatting.Output" object enables the creation and formatting of text according to the parameters passed.

Methods

Methods	Description
`format(value, pattern [, lcid])`	Takes any number or random text and formats it according to the parameters passed. The language can optionally be specified via the LCID. When the LCID is not transferred, the current language is used.

Example

```
var floatValue = WebCC.Extensions.Formatting.Output.format( 42.1111111, '{F2}', 'de-DE' );
//Ergebnis: 42,11
var hexValue = WebCC.Extensions.Formatting.Output.format( 45054, '{H,2}' );
//Ergebnis: AF FE
var dateValue = WebCC.Extensions.Formatting.Output.format( 1609745948315, '{D,@yyyy/MM/dd}
{T,@HH:mm:ss}');
//Ergebnis: 2021/01/04 07:39:08
```

See also

Basics of extensions (Page 7804)

20.2.5.4 Dialog extension

The dialog extension enables you to open a dialog window that is not limited by the display range of the Control. A URL is made available to the user interface; it is displayed in the dialog. A JSON data model is also made available. It is transferred to the dialog and returned as soon as the dialog is closed.

The use of dialogs is asynchronous.

"WebCC.Extensions.Dialogues" object

The "WebCC.Extensions.Dialogues" object allows for the creation and use of one or more dialogs.

The object has the following methods and properties:

Methods	Description
`create(id, view, data, [options])`	Creates a "Dialog" object. The "id" parameter is transferred as a string. The "view" parameter is transferred as a string that contains a relative URL. The "data" parameter contains random data this is transferred to the dialog. The optional "options" parameter contains a JSON object with the following options: • Information on the size of the dialog box: "minwidth", "maxwidth", "minheight", "maxheight" and "unit". • A Boolean value "resizable" that defines whether the size of the dialog box can be changed. • A "caption" string that is displayed in the title of the dialog.
`list()`	Returns a string array that contains the IDs of currently created and opened dialog boxes. As soon as the dialog box is closed, the ID is no longer displayed in the string array.
`get(id)`	Returns a "Dialog" object with the transferred ID. If the ID does not exist, ZERO is returned.

Property	Description
`self`	The "self" property contains the "Dialog" object.

"WebCC.Extensions.Dialog" prototype

The "WebCC.Extensions.Dialog" prototype enables the interaction with the dialog instance and offers the following methods and events:

Methods	Description
`open(width, height)`	Creates a dialog and returns a "promise" property. If the dialog is already open, only the "promise" property is returned. The dialog is displayed in the current screen with the specified width and height.
`close([result])`	Closes a dialog and fulfills the associated "promise" property. The "result" parameter is optional and can be of any type. The "result" parameter normally contains the data that were transferred when the dialog was opened and were subsequently changed.
`cancel([reason])`	Closes a dialog. The "promise" property is rejected. The "reason" parameter is optional and can be of any type.

Property	Description
`id`	The "id" property contains the ID of the dialog. The ID is available as a string.
`promise`	The "promise" property contains information about the status of a dialog.
`data`	The "data" property contains the data that were transferred when a dialog was created.

Example: Opening a dialog

```
var dialog =  WebCC.Extensions.Dialogues.create('id_1','input.html',
'myData',
{ caption: 'GaugeValue', resizeable: false } );
if ( dialog ) {
  // open( width, height ) returns promise
  dialog.open( 250, 150 ).then(
    function success( data ) {
      // close called
      WebCC.Properties.GaugeValue = data;
      updateValue( data );
  }).catch(
    function ( reason ) {
      // cancel called
  });
}
```

Example: Access to data within a dialog

```
var self = WebCC.Extensions.Dialogues.self;
if ( self ) {
  // initialize dialog data
  var inputValue = self.data;
  // 'myData'
}
```

Example: Closing a dialog with fulfilled promise

```
var self = WebCC.Extensions.Dialogues.self;
if ( self ) {
  // close([result]) fullfill dialog promise
  self.close( data );
}
```

Example: Closing a dialog in case of cancellation

```
var self = WebCC.Extensions.Dialogues.self;
if ( self ) {
  // cancel([reason]) reject dialog promise
  self.cancel();
}
```

See also

Basics of extensions (Page 7804)

20.2.6 Revision of a graphical user interface

Introduction

User interfaces can be used as Custom Web Control by using the framework. This document is intended to describe the process by means of an example and uses a provided user interface. With this user interface, a slider controls the movement of a pointer.

The user interface can be found in the application example at the following address: SIOS entry (https://support.industry.siemens.com/cs/ww/en/view/109779176)

The code examples shown in this section can be found in the "index.html" file of the application example.

Conversion of the color coding

The TIA Portal and the manifest file use different color coding. The manifest file is not able to work with hexadecimal values, but only accepts decimal values. The author of the manifest file must convert the hexadecimal values as defined in the TIA Portal into decimal values.

To use a web page as a Custom Web Control, the encoding must be converted.

Example: Color coding

```
function toColor(num) {
  num >>>= 0;
  var b = num & 0xFF,
      g = (num & 0xFF00) >>> 8,
      r = (num & 0xFF0000) >>> 16,
      a = ((num & 0xFF000000) >>> 24) / 255;
  return 'rgba(' + [r, g, b, a].join(',') + ')';
}
```

Defining the default values of properties

Default properties are defined in the TIA Portal and for Runtime projects. These default properties define, among other things, font size, line thickness and value ranges. Default properties are obtained in Runtime during initialization of the setup.

Example: Defining default values

```
var defaultProperties = {
  GaugeValue: 20,
  GaugeBackColor: 4294967295,
  Alignment:
  {
    Vertical: 'Center'
  },
  LineThickness: 20,
  FontSize: 16,
  MinValue: 0,
  MaxValue: 50,
  DivisionCount: 5,
  Zones: [
    { Min: 0, Max: 30, StrokeColor: 4281381677 },
    { Min: 30, Max: 40, StrokeColor: 4294958336 },
    { Min: 40, Max: 50, StrokeColor: 4293934654 }
  ]
}
```

Initialization of the Custom Web Control

To be able to function as a Custom Web Control within WinCC, the "WebCC" object must be initialized. After that the Control can be created.

Initialization takes place in "index.html".

Initialization of the Custom Web Control

```
WebCC.start(
  // callback function; occurs when the connection is done or failed.
  // "result" is a boolean defining if the connection was successfull or
not.
  function (result) {
    if (result) {
      console.log('connected successfully');
      initializeGauge();
      // Set current values
      setProperty({ key: 'GaugeBackColor', value:
WebCC.Properties.GaugeBackColor });
      setProperty({ key: 'Alignment', value: WebCC.Properties.Alignment });
      setProperty({ key: 'LineThickness', value:
WebCC.Properties.LineThickness });
      setProperty({ key: 'DivisionCount', value:
WebCC.Properties.DivisionCount });
      setProperty({ key: 'FontSize', value: WebCC.Properties.FontSize });
      setProperty({ key: 'Zones', value: WebCC.Properties.Zones });
      setProperty({ key: 'MaxValue', value: WebCC.Properties.MaxValue });
      setProperty({ key: 'MinValue', value: WebCC.Properties.MinValue });
      setProperty({ key: 'GaugeValue', value:
WebCC.Properties.GaugeValue });
      // Subscribe for value changes
      WebCC.onPropertyChanged.subscribe(setProperty);
    }
    else {
      console.log('connection failed');
    }
  },
  // contract (see also manifest.json)
  {
    // Methods
    methods: {
     },
    // Events
    events: {
    },
    //Properties
    //////////
    properties: defaultProperties
  },
  // placeholder to include additional Unified dependencies (not used in
this example) [],
  // connection timeout
  10000
);
```

Operating the Control via WinCC

To operate the Custom Web Control via WinCC, some functions need to be implemented. The functions show an example of the use of the API object. The sequences of the functions are specific to this example, so they are not explained in detail.

Example: Operating the Control

```
// Updates the value shown by the gauge whenever it is changed, e.g. by a
WinCC Unified tag or script.
// This function will be called by "setProperty" whenever the contract
property GaugeValue is changed.
// - value: number that contains the new value to be shown in the gauge
meter. function updateValue(value) {
  gauge.set(value);
  const newZoneIndex = gauge.options.staticZones.indexOf(
    gauge.options.staticZones.
      filter(zone => zone.min <= gauge.value && gauge.value <=
zone.max).pop()
  );
  if (newZoneIndex != currentZoneIndex) {
    currentZoneIndex = newZoneIndex;
    WebCC.Events.fire('ZoneChanged', newZoneIndex);
  }
}

// Updates the alignment of the whole gauge inside the control. You can
place it at the top, middle or bottom.
// This function will be called by "setProperty" whenever the user changes
the alignment.
// - alignment: object that contains an enum property "Vertical" that can be
either "Top", "Center" or "Bottom".
function updateAlignment(alignment) {
  const item = document.getElementById('gauge');
  let vertVal = '0';
  let topVal = '0';
  switch (alignment.Vertical) {
    case 'Top':
      break;
    case 'Center':
      topVal = '50%';
      vertVal = '-50%';
      break;
    case 'Bottom':
      topVal = 'inherit';
      break;
  }
  item.style.top = topVal;
  item.style.transform = 'translate(0,' + vertVal + ')';
}

// Updates the labels of the gauge. All labels have to be updated whenever
the DivisionCount, MaxValue, MinValue or FontSize is changed.
// This function will be called by "setProperty" whenever one of those
contract properties change.
function updateLabels() {
  const labels = new Array(.Properties.DivisionCount).fill(0).map(
    (x, i) => (i + 1) * (WebCC.Properties.MaxValue -
WebCC.Properties.MinValue/WebCC.Properties.DivisionCount +
WebCC.Properties.MinValue
```

```
  );
  labels.unshift(WebCC.Properties.MinValue);
  gauge.setOptions({
    staticLabels: {
      font: WebCC.Properties.FontSize + 'px "Siemens Sans"',
      labels: labels
    }
  });
}

// Paints the given zones inside the gauge. This function will be called by
"setProperty" whenever a zone is changed or
// zones will be added or removed.
// - zones: array of new zones to be painted
function updateZones(zones) {
  gauge.setOptions({
    staticZones: zones.map(item => {
      return { strokeStyle: toColor(item.StrokeColor), min: item.Min, max:
item.Max };
    })
  });
}

// This is a callback function that is called every time a contract property
changes. The function forwards the change to
// other functions so you can see the new value in the control.
// - data: object containing a key and a value property. The "key" contains
the name of the changed contract property and the "value" contains the new
value.
function setProperty(data) {
  // console.log('onPropertyChanged ' + data.key); // uncomment this line
to check whether data is incoming in the browser console from WinCC Unified
  switch (data.key) {
  case 'GaugeValue':
    updateValue(data.value);
    break;
  case 'GaugeBackColor':
    document.body.style.backgroundColor = toColor(data.value);
    break;
  case 'Alignment':
    updateAlignment(data.value);
    break;
  case 'LineThickness':
    gauge.setOptions({ lineWidth: data.value / 100 });
    break;
  case 'FontSize':
    updateLabels();
    break;
  case 'MinValue':
    gauge.setMinValue(data.value);
    updateLabels();
    break;
  case 'MaxValue':
    gauge.maxValue = data.value;
    updateLabels();
```

```
    break;
  case 'DivisionCount':
    updateLabels();
    break;
  case 'Zones':
    updateZones(data.value);
    break;
  }
}

// Let the given zone blink by descreasing and increasing the alpha value of
the zone color from 0% to 100% and back to original value 2 times.
// - zoneIndex: integer as index of the zone that will blink.
function blinkZone(zoneIndex) {
  const currentZone = gauge.options.staticZones[zoneIndex];
  const rgba = currentZone.strokeStyle.split(',');
  const originalRgba = Number(rgba[3].replace(')', ''));
  let currentRgba = originalRgba;
  let state = 0; // 0: falling, 1: raising, 2: falling again
  let currentRound = 0;
  const timerId = setInterval(() => {
    switch (state) {
      case 0:
        currentRgba -= 0.2;
        if (currentRgba <= 0) {
          currentRgba = 0;
          state = 1;
        }
        break;
      case 1:
        currentRgba += 0.2;
        if (currentRgba >= 1) {
          currentRgba = 1;
          state = 2;
        }
        break;
      case 2:
        currentRgba -= 0.2;
        if (currentRgba < originalRgba) {
          currentRound++;
          if (currentRound >= 2) {
            clearInterval(timerId);
            return;
          } else {
            currentRgba = originalRgba;
            state = 0;
          }
        }
        break;
    }  rgba[3] = currentRgba.toFixed(1);
    currentZone.strokeStyle = rgba.join(',') + ')';
    gauge.setOptions(gauge.options.staticZones);
  }, 50);
}
```

20.2.7 Creating the ZIP file

To use the Custom Web Control, the hierarchy of folders and files must be compressed. The data must be available in compressed form. To create the file, you can use any application that can generate a valid file with the extension ".zip".

The name of the ZIP file must match the GUID, for example, "{551BF148-2F0D-4293-8E10-C9C3A1A6A073}.zip".

Note

GUID generation

For example, a GUID can be created as follows:

- In Visual Studio under "Tools > Create GUID"
- Under https://www.guidgenerator.com/ (https://www.guidgenerator.com/)

20.2.8 Restrictions

Unified Comfort Panel

When you make the Custom Web Control available on a Unified Comfort Panel, note the following special features:

- The Custom Web Control is executed locally without being hosted in a web server.
- Links outside of the Custom Web Control are not supported, e.g. http links.
- The retrieval of external data or adding of contents is not supported, for example, via the "XMLHttpRequest" object or "fetch".
- Debugging of the Custom Web Control on the Unified Comfort Panel is not supported.

Tips for an efficient procedure

If you want to test whether your Custom Web Control can be executed on the Unified Comfort Panel, run the file "Index.html" locally on a PC without hosting the Custom Web Control in its own web server.

Rounding error

The JavaScript data type "Number" is a 64-bit floating point data type. For integer values, a secure display of up to 15 digits is possible before rounding errors occur.

Tags that use "DInt" or "Date", for example, can therefore lead to rounding errors when used in Custom Web Controls.

To avoid rounding errors, use the prototypes "Big" and "DatePrecise".

Logic operation of complex data types

When using complex types, such as an array or user data types, only the elements of the bottom level can be linked to a property.

Access of devices outside of the network

When the custom web control is accessed by a device outside of the network, it may not be possible to display or operate the Custom Web Control. In this case, access to the Custom Web Control depends on the network and security settings.

20.2.9 Installing and using Custom Web Controls

Custom Web Controls are freely-programmable and serve as a specific solution that goes beyond the functionalities of the toolbox provided. You can use Custom Web Controls like any other tool in the screens.

Requirement

- A project has been created.
- An HMI device has been created.
- A screen has been created.

Installing Custom Web Control

To install Custom Web Controls for a TIA Portal project, follow these steps:

1. Open the directory of your project.
2. Open the "UserFiles" subfolder.
3. Create a folder with the name "CustomControls".
4. Store the created program as *.zip file in the "CustomControls" folder.
5. In the TIA Portal, click on the "Update" button in the "Toolbox" > "My Controls" task card. The Custom Web Control is displayed in the "Toolbox" task card of the "Screens" editor.

Using Custom Web Control

1. Drag the Custom Web Control from the "Toolbox" > "My Controls" task card onto the screen.
2. Select the Control.
3. In the Inspector window, go to "Properties > Events".
4. Configure system functions or scripts for the events used in the Control.
5. In the Inspector window, go to "Properties > Properties > Interface".
6. Assign static values for the interface properties or dynamize the interface properties according your requirements.
   When dynamizing with tags, note that the Control's access to the tags is "Read only" by default.
   If you want the Control to change the values of the tags, remove the check mark.
7. Compile and load the Runtime project.

20.2.10 Updating Custom Web Controls

Introduction

After changing the properties for custom web controls or configuring new events, you can update custom web controls.

Updating custom web controls

If you have changed the properties for custom web controls or configured new events, you can update the custom web controls as follows:

1. Click the Update icon in the "My Controls" palette.
2. Custom web controls are updated.
3. A message appears in the Inspector window:
   - "The object '{0}' was updated successfully".
   - "The object '{0}' was updated successfully, but some properties have been lost due to incompatible changes to the interface."
   - "All objects are up to date". You have not changed any properties or alarms.

Note

The placeholder '{0}' stands for a unique and complete path on which the custom web control is stored.

Restrictions for the update

If you have opened a project as read-only, the update is not possible.

The following changes to the custom web control prevent the automatic update:

- Renaming properties or events
- Deleting properties or events
- Changing the data type

20.3 Runtime API

20.3.1 Basics

Task of the runtime API

Runtime API describes the open programming interface of WinCC Unified PC RT. With the Runtime API, you can use the internal functions of WinCC in your own applications. With an ODK client, you can read out all the objects of the runtime system and change their runtime attributes, for example, for tags or alarms.

The ODK is optimized for the processing of mass data when special objects are used, for example the reading or writing of 1000 tags in one pass.

Note

Siemens is not liable for and does guarantee the compatibility of the data and information transported via the API interfaces with third-party software.

We expressly point out that improper use of the API interface can result in data loss or production downtimes.

Requirement

- Programming environment is installed, e.g. MS Visual Studio
- WinCC Runtime Unified Scada RT is installed.

Application of C++ and .NET

The Runtime API makes all the interfaces available for the access to the runtime system in the languages C++ and C#.

Name-based addressing of objects

The objects of the Runtime system are addressed by their name and the full name path.

The name path of objects consists of several components and has the following syntax:

```
[SystemName::][ObjectName][.ElementPath][:SubElementName]
```

- `SystemName`
  Name of a Runtime systems (optional)
  If the "SystemName" is omitted, the object is searched for on the local runtime system.
- `ObjectName`
  Name of a tag or a structure

- `ElementPath`
  Element of a structure
- `SubElementName`
  Subelement of an object, e.g. alarm or logging tag of a tag.

Examples for access to different object types:

- Simple tag: `MyRTSystem::MySimpleTag`
- Structure tag: `MyRTSystem::Motor.Temperature`
- Alarm of a simple tag: `MyDiscreteTag:MyDiscreteAlarm`
- Alarm of a structure tag: `Motor.Temperature:MyAnalogAlarm`
- Logging tag: `MySimpleTag:MyLoggingTag`
- Connection: `MyRTSystem::MyHmiConnection`

20.3.2 Creating a minimal ODK client

Introduction

An ODK client uses the ODK API to access objects of the WinCC Unified system.

In the following, an ODK client is created for use of the Runtime API in the C# and C++ languages.

The programs only contain the most needed components of a simple client. They form the framework for all the subsequent runtime code examples in this documentation. See also section Code samples (Page 7832).

Note

You will find additional programming examples on the installation medium in the file "Support\Openness\Siemens.Unified.Openness_SDK_<version number>.zip" in the subdirectory"ODK\samples".

Requirement

- Development environment is installed.
- The ODK SDK was extracted locally on your computer. You will find the ODK SDK in the "Support\Openness" folder on the WinCC Unified DVD in the file "Siemens.Unified.Openness_SDK_<version number>.zip".

  Note

  If you create a C++ ODK client, you must set the system tag "PATH=C:\Program Files\Siemens\Automation\WinCCUnified\bin" and perform a restart.

Procedure C# client

1. Create a new .NET project in the development environment.
2. Carry out the following project settings:
   - Target framework is .NET 4.6.
   - ODK client is "Release" version for the x64 platform.
3. Create references to the following assembly: Siemens.Runtime.HmiUnified.Interfaces.dll (Copy Local = False)
   You will find the assembly in the local folder to which you have extracted Openness_SDK.zip, in the subfolder "ODK\bin".
4. Create a program with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Siemens.Runtime.HmiUnified;

namespace Siemens.Runtime.HmiUnified.TestClient
{
    class Program
    {

        static void Main(string[] args)
        {
            try
            {
                using (IRuntime runtime = Runtime.Connect())
                {
                    //do runtime operations
                }
            }
            catch (Exception ex)
            {
                System.Console.WriteLine(string.Format("Exception occured
{0}", ex.Message));
            }
        }
    }
}
```

Procedure C++ client

1. Create a new C++ project in the development environment.
2. Set the following include directories for required headers:
   - <Local folder to which you have extracted the ODK SDK>\ODK\include\ODK
   - <Local folder to which you have extracted the ODK SDK>\ODK\include\ODK\include\CF

3. Create references to the following libraries in "<Local folder to which you have extracted the ODK SDK>\ODK\include\ODK\lib":
   - HmiUnifiedRt.lib
   - CfCore.lib
4. Create a reference to the following directory as "Additional Library Directory":
   - <Local folder to which you have extracted the ODK SDK>\ODK\lib
5. Create a program with the following code:

```
#include <CfTL>

#include "IOdkRt.h"
#include "IOdkRtTag.h"
#include "IOdkRtTagLogging.h"
#include "IOdkRtAlarm.h"
#include "IOdkRtAlarmLogging.h"
#include "IOdkRtCpm.h"
#include "IOdkRtConnection.h"
#include "IOdkRtUmc.h"

#include <stdio.h>
#include <tchar.h>
#include <iostream>

using namespace Siemens::Runtime::HmiUnified;
using namespace Siemens::Runtime::HmiUnified::Common;
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    CCfString projectName = L"";
    IRuntimePtr pRuntime;

    if(CF_SUCCEEDED(Connect(projectName, &pRuntime)
    {
        // do runtime operations here
    }
    return 0;
}
```

Result

The program core of an ODK client is created.

You can complete the program with the fragments from the following code examples for ODK-API.

20.3.3 Authorizing users

Introduction

Before the ODK client can be used, all users that run the ODK client must be authorized.

Note

Only authenticated users can access the data of a project over the Runtime API with an ODK client.

Requirement

WinCC Unified RT setup has been carried out completely on the Runtime computer.

Procedure

Add the user who corresponds to the logged-on Windows user on the Runtime computer to the Windows user group "SIMATIC HMI" in the Windows user management.

Result

When the ODK client is connected to the Runtime system, the logged-on Windows user is authenticated via the user management of the Runtime computer.

If one of the checks fails, the ODK client does not establish a connection (error: "Authentication error" or "User has no access right").

20.3.4 Startup and shutdown behavior of an ODK application

20.3.4.1 Autostart of an ODK application

You have the possibility of starting ODK applications automatically on start-up of the device.

Requirement

Runtime is configured in such a way that it automatically started on start-up of the device without a user having to be logged on. (Default setting)

Procedure

In the Windows Task Scheduler, create a task which starts the ODK application on start-up of the device.

Note

User Service Mode

To use the ODK application in Service Mode, configure the security options in the dialog "Create task" in such a way that a user does not have to be logged on for starting the task.

Under Windows 10 activate the option "Run whether user is logged on or not".

Result

The "Connect" method of IRuntime wait for a maximum of ten minutes after the start of the ODK application until the Runtime has started up.

20.3.4.2 Shutdown behavior

You have the option to be notified by the system on shutdown of Runtime, for example, to start cleanup work on the client.

For this purpose, subscribe the system tag "@SystemActivationState" for monitoring. "@SystemActivationState" signals whether Runtime is active and can have the following values:

- System startup in progress (1)
- System started (activated) (2)
- System stopped (3)
- System shutdown in progress (4)
- System restart in progress (5)

Value 4 is the trigger to start cleanup work.

Note

Interface calls on shutdown

Do not call any functions of the ODK interfaces while the system is shut down.

20.3.4.3 Restart behavior

Tags subscribed for monitoring

After Runtime is restarted, you can continue to use the existing subscriptions.

20.3.5 Syntax of the alarm filter

With an AlarmSubscription, a filter can be transferred so that not all active alarms of the alarm system are notified, but only those which match the filter. The filter syntax is based on SQL syntax. However, only the WHERE instruction is relevant. The keyword "WHERE" must be omitted.

Operators

The following operators can be used in the filter string of the alarm filter:

Operator	Description	Example
`=`	equal to	`Name = 'Recipe246'`
`<>`	not equal	`Value <> 0.0`
`>`	greater than	`Value > 25.0`
`<`	less than	`Value < 75.0`
`>=`	greater than or equal to	`Value >= 25.0`
`<=`	less than or equal to	`Value <= 75.0`
`OR, ||`	logical OR	`State = 1 OR State = 3`
`AND, &&`	logical AND	`Value >= 25.0 AND Value <= 75.0`
`BETWEEN`	within a range	`Value BETWEEN 25.0 AND 75.0`
`NOT BETWEEN`	outside a range	`Value NOT BETWEEN 25.0 AND 75.0`
`LIKE string`	corresponds to the string *string*	`Name LIKE 'Motor*'`
`NOT LIKE string`	does not correspond to the string *string*	`Name NOT LIKE 'Valve*'`
`IN (v1, v2, …)`	corresponds to one or more values	`State IN (1, 4, 7)`
`NOT IN (v1, v2, …)`	does not correspond to one or more values	`State NOT IN (0, 2, 3, 5, 6)`
`(…)`	brackets expressions	`Value <= 75.0 AND (State = 1 OR State = 3)`

Precedence of the operators:

Rank	Operators
1	• Relational operators: =, <>, >, <, >=, <= • LIKE • IN • BETWEEN
2	NOT
3	AND, &&
4	OR, ||

Permitted wildcards:

Wildcard	Description	Example
*	Replaces 0 to more characters	`Name LIKE 'Motor*'` `Reference = <1.*.15>1`
?	Replaces 1 character	`Name = 'Recipe?'`

See also

IAlarmSubscription (Page 7895)

IPlantObjectAlarmSubscription (Page 7937)

IPlantObjectAlarmSubscription (Page 8129)

IAlarmSubscription (Page 8077)

20.3.6 Locale IDs of the supported languages

At the AlarmSubscription, there is a Language property which defines the language of the alarm filter and the language of the alarm texts. In this case, a locale ID from the table below must be entered.

The following table contains the Microsoft locale IDs of the languages supported in the TIA Portal:

Language	Country/Region	Locale ID
Afrikaans	South Africa	1078
Albanian	Albania	1052
Armenian	Armenia	1067
Azerbaijani (Cyrillic)	Azerbaijan	2092
Azerbaijani (Latin)	Azerbaijan	1068
Basque	Basque country	1069
Belarusian	Belarus	1059
Bulgarian	Bulgaria	1026
Chinese	Chinese (Hong Kong S.A.R.)	3076
Chinese	Chinese (Macao S.A.R.)	5124
Chinese	Chinese (Singapore)	4100
Chinese	Chinese (Taiwan)	1028
Chinese	Chinese (PR China)	2052
Danish	Denmark	1030
German	Germany	1031
German	Liechtenstein	5127
German	Luxembourg	4103
German	Austria	3079
German	Switzerland	2055
English	Australia	3081

Language	Country/Region	Locale ID
English	Belize	10249
English	United Kingdom	2057
English	Ireland	6153
English	Jamaica	8201
English	Canada	4105
English	Caribbean	9225
English	New Zealand	5129
English	Philippines	13321
English	Zimbabwe	12297
English	South Africa	7177
English	Trinidad and Tobago	11273
English	USA	1033
Estonian	Estonia	1061
Faroese	Faroe Islands	1080
Finnish	Finland	1035
French	Belgium	2060
French	France	1036
French	Canada	3084
French	Luxembourg	5132
French	Monaco	6156
French	Switzerland	4108
Galician	Galicia	1110
Georgian	Georgia	1079
Greek	Greece	1032
Hindi	India	1081
Indonesian	Indonesia	1057
Icelandic	Iceland	1039
Italian	Italy	1040
Italian	Switzerland	2064
Japanese	Japan	1041
Kazakh	Kazakhstan	1087
Catalan	Catalonia	1027
Kyrgyz	Kyrgyzstan	1088
Konkani	India	1111
Korean	Korea	1042
Croatian	Croatia	1050
Latvian	Latvia	1062
Malay	Brunei Darussalam	2110
Malay	Malaysia	1086
Macedonian	Macedonia, FYRM	1071
Mongolian (Cyrillic)	Mongolia	1104
Dutch	Belgium	2067
Dutch	Netherlands	1043

Language	Country/Region	Locale ID
Norwegian (Bokmal)	Norway	1044
Norwegian (Nynorsk)	Norway	2068
Polish	Poland	1045
Portuguese	Brazil	1046
Portuguese	Portugal	2070
Romanian	Romania	1048
Russian	Russia	1049
Sanskrit	India	1103
Swedish	Finland	2077
Swedish	Sweden	1053
Serbian (Cyrillic)	Serbia and Montenegro (formerly)	3098
Serbian (Latin)	Serbia and Montenegro (formerly)	2074
Slovakian	Slovakia	1051
Slovenian	Slovenia	1060
Spanish	Argentina	11274
Spanish	Bolivia	16394
Spanish	Chile	13322
Spanish	Costa Rica	5130
Spanish	Dominican Republic	7178
Spanish	Ecuador	12298
Spanish	El Salvador	17418
Spanish	Guatemala	4106
Spanish	Honduras	18442
Spanish	Columbia	9226
Spanish	Mexico	2058
Spanish	Nicaragua	19466
Spanish	Panama	6154
Spanish	Paraguay	15370
Spanish	Peru	10250
Spanish	Puerto Rico	20490
Spanish	Spain	3082
Spanish	Uruguay	14346
Spanish	Venezuela	8202
Swahili	Kenya	1089
Tatar	Russia	1092
Thai	Thailand	1054
Czech	Czech Republic	1029
Turkish	Turkey	1055
Ukrainian	Ukraine	1058
Hungarian	Hungary	1038
Uzbek (Cyrillic)	Uzbekistan	2115

Language	Country/Region	Locale ID
Uzbek (Latin)	Uzbekistan	1091
Vietnamese	Vietnam	1066

20.3.7 Code samples

ODK is supplied with code samples for using the Runtime interfaces. Open the local folder to which you have extracted the file "Support\Openness\Siemens.Unified.Openness_SDK_<version number>.zip".

You will find the code samples in the subfolder "\ODK\samples".

Using the code samples in the help

To reduce the complexity of the code samples and enable better readability, the examples in the help deliberately exclude troubleshooting and freeing up memory.

The application programmer must add these elements during programming.

Constructs affected under C#

- Exception handling with try...catch...finally

```
try
{
   …
}
catch (Exception ex)
{
   …
}
finally
{
}
```

- Freeing up memory with Dispose or using (...)
  See also section Releasing objects (Page 7833).

For a description of how to evaluate ODK-specific errors, see section Error-handling interfaces (Page 7841).

Constructs affected under C++

- Methods for outputting error information
  For an example of implementation of the printErrorInformation method, see section IErrorInfo (Page 8001).
- Freeing up allocated memory
- Null pointer check: `if (pObject != nullptr) {…}`
- Error code check: `if (CF_SUCCEEDED(errorCode) {…}`

For a description of how to evaluate ODK-specific errors, see section Error codes of the C++ interfaces (Page 7986).

20.3.8 Description of the C# interfaces

20.3.8.1 Releasing objects

Creating objects with GetObject

In .NET-ODK, you create the objects with the "GetObject" method, for example:

```
ITagSet odkTagSet = runtime.GetObject<ITagSet>();
```

Memory is created internally for the object. In .NET, the Garbage Collector automatically releases the memory when an object is no longer needed. However, the memory is released at an indefinite time.

Note

The indefinite execution of the Garbage Collector may cause the memory to increase and appear as if it is not being released again. The real reason for this is that the Garbage Collector has not yet started!

Releasing objects created with GetObject

The ODK client should release the memory of objects created using the "GetObject" method as soon as the object is no longer needed.

The following cases must be distinguished here:

- Using the objects by synchronous ODK methods
- Using the objects by asynchronous ODK methods

Example when using synchronous ODK methods

When releasing objects used by synchronous ODK methods, use the keyword `using`:

Copy code

```
try
{
    using (ITag myTag = runtime.GetObject<ITag>("Tag1"))
    {
        IProcessValue value = myTag.Read(HmiReadType.Cache);   // Reads synchronous
    }
}
catch (OdkException ex)
{
    System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
}
```

Example when using asynchronous ODK methods

When you release objects that are used by asynchronous ODK methods, you can use the "Dispose" method. This can be called in the callback method:

Copy code

```
ITagSet odkTagSet = runtime.GetObject<ITagSet>();
odkTagSet.Add(new string[] { "Tag1", "Tag2" });

// Assign callback function
odkTagSet.OnReadResult += odkTagSet_OnReadResult;
odkTagSet.ReadAsync();// Reads asynchronous


void odkTagSet_OnReadResult(ITagSet sender, IList<IProcessValue> values, bool completed)
{
    try
    {
        …
    }
    catch (OdkException ex)
    {
        …
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose(); // Release memory
        }
    }
}
```

20.3.8.2 Interfaces of the Runtime environment

IRuntime

Description

The C# interface "IRuntime" specifies properties and methods for handling the Runtime system. The "Connect" method of the "Runtime" class is called to establish the connection to the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members "Runtime"

The class implements the following method:

"Connect" method

Checks whether valid licenses are available for the products installed on the Runtime device:

- Yes: Connects ODK application and Runtime project.
- No: Supplies an error code. Use the interfaces for error handling to query the error description (license missing, expired, etc.).

The method is overloaded:

- Connect to locally running Runtime project. Logged-on Windows user is authenticated.
  `IRuntime Connect()`
- Connect to locally running Runtime project. The logged-on Windows user is not authenticated; instead, the user is specified as a parameter.

  Note

  Can only be used in a future version!

  `IRuntime Connect(string userName, string password)`

  – `user`
    User name

  – `password`
    Password

- Connect to a specific Runtime project. Logged-on Windows user is authenticated.

  Note

  Can only be used in a future version!

  `IRuntime Connect(string value)`
  `value`
  Name of a Runtime project

- Connect to a specific Runtime project. The logged-on Windows user is not authenticated; instead, the user is specified as a parameter.

  Note

  Can only be used in a future version!

  `IRuntime Connect(string value, string userName, string password)`

  – `value`
    Name of a Runtime project

  – `user`
    User name

  – `password`
    Password

"Dispose" method

Enable Runtime system with all resources.

`void Dispose()`

Members "IRuntime"

The following properties and methods are specified in the interface:

"ProjectName" property

Name of the current project

Note

Can only be used in a future version!

```
string ProjectName { get; }
```

"UserName" property

Name of the logged-on user

```
string UserName { get; }
```

"Product" property

Return version information and installed options of the Runtime system as "IProduct" object.

```
IProduct Product { get; }
```

"GetObject" method

Create a new instance of an object type T in a project.

```
T GetObject<T>(params object[] parameters)
```

The object type T adopts the following values:

- `ITag`, `ITagSet` or `ITagSetQCD`
  Access to tags
- `IAlarm`, `IAlarmSet` or `IAlarmSubscription`
  Access to alarm system
- `ILoggedTag` or `ILoggedTagSet`
  Access to logging tags
- `IAlsrmLogging` or `IAlarmLoggingSubscription`
  Access to logged alarms
- `IUserManagement`
  Access to user management
- `IConnection` or `IConnectionSet`
  Access to connections

`parameters`
Optional: A name or array with names of objects of the respective object type

"GetOption" method

Return an installed option of the Runtime system as "IOption" object using the name.

```
IOption GetOption(string optionName)
```

`optionName`
Name of the installed option

Example

Initialize the ODK and establish a connection to the active project of the Runtime system.

Copy code

```
public static IRuntime runtime = null;

public void Connect()
{
    // Connect to running project
    runtime = Siemens.Runtime.HmiUnified.Runtime.Connect();
}
```

Initialize an "IProduct" object and output the technical product version of the Runtime system:

```
public void GetVersionInfo(IRuntime runtime)
{
    IProduct product = runtime.Product;
    IVersionInfo version = product.Version;
    System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
version.Major, version.Minor, version.ServicePack, version.Update));

    ...
}
```

Access a tag with the name "Tag1":

```
public void ReadSingleTagSync()
{
    ITag myTag = runtime.GetObject<ITag>("Tag1");
   //further tag processing
}
```

See also

IProduct (Page 7837)

IErrorResult (Page 7841)

IProduct

Description

The C# interface "IProduct" specifies properties for handling product information of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties are specified in the interface:

"Options" property

Return installed options of the Runtime system as a list of "IOption" objects.

```
IList<IOption> Options { get; }
```

"Version" property

Return version structure of the Runtime system as "IVersionInfo" object.

```
IVersionInfo Version { get; }
```

Example

The version of the "IProduct" object is read out and iterated via the installed "IOption" objects:

Copy code

```
public void GetVersionInfo(IRuntime runtime)
{
   // Get product version
   IProduct product = runtime.Product;
   IVersionInfo version = product.Version;
   System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
version.Major, version.Minor, version.ServicePack, version.Update));
   if (product.Options.Count > 0)
   {
      foreach (IOption op in product.Options)
      {
         IVersionInfo opVersion = op.Version;
         // Iterate through options and get version
         System.Console.WriteLine(string.Format("Option name: {0}", op.Name));
         System.Console.WriteLine(string.Format("Option version: {0}.{1}.{2}.{3}",
opVersion.Major, opVersion.Minor, opVersion.ServicePack, opVersion.Update));
      }
   }
}
```

See also

IRuntime (Page 7834)

IOption (Page 7839)

IVersionInfo (Page 7840)

IOption

Description

The C# interface "IOption" specifies properties and methods for handling installed product options of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of an installed option of the Runtime system

```
String Name { get; }
```

"Version" property

Return version structure of an installed option of the Runtime system as "IVersionInfo" object.

```
IVersionInfo Version { get; }
```

"GetObject" method

Create new instance of an object type T of the option.

```
T GetObject<T>(params object[] parameters)
```

- `T`
  The value defines a specific object type of the option.
- `parameters`
  Optional: A parameter or array with parameters for the object type of the option

Example

Instantiate and use installed options with name "MyOptionName":

Copy code

```
public void GetOptionObject()
{
   //load option component by name
   IMyOption rtOption = (IMyOption)runtime.GetOption("MyOptionName");
   //create a instance of the option object IMyOptionObject
   IMyOptionObject optionObject = rtOption.GetObject<IMyOptionObject>();
   string strMethod = optionObject.MyMethod();
   string strProperty = optionObject.MyProperty;
}
```

See also

IProduct (Page 7837)

IVersionInfo (Page 7840)

IVersionInfo

Description

The C# interface "IVersionInfo" specifies properties for handling version information of the Runtime system.

Members

The following properties are specified in the interface:

"Major" property

Main version of the Runtime system or of an installed option

```
uint16 Major { get; }
```

"Minor" property

Minor version of the Runtime system or of an installed option

```
uint16 Minor { get; }
```

"ServicePack" property

Service pack of the Runtime system or of an installed option

```
uint16 ServicePack { get; }
```

"Update" property

Update version of the Runtime system or of an installed option

```
uint16 Update { get; }
```

Example

Information about the installed "IOption" options of the runtime system is output:

Copy code

```
public void GetVersionInfo(IRuntime runtime)
{
   // Get product version
   IProduct product = runtime.Product;
   IVersionInfo version = product.Version;
   System.Console.WriteLine(string.Format("Product version: {0}.{1}.{2}.{3}",
version.Major, version.Minor, version.ServicePack, version.Update));
   if (product.Options.Count > 0)
   {
      foreach (IOption op in product.Options)
      {
         IVersionInfo opVersion = op.Version;
         // Iterate through options and get version
         System.Console.WriteLine(string.Format("Option name: {0}", op.Name));
         System.Console.WriteLine(string.Format("Option version: {0}.{1}.{2}.{3}",
opVersion.Major, opVersion.Minor, opVersion.ServicePack, opVersion.Update));
      }
   }
}
```

See also

IProduct (Page 7837)

IOption (Page 7839)

20.3.8.3 Error-handling interfaces

IErrorResult

Description

The C# interface "IErrorResult" specifies properties of error results in runtime.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties are specified in the interface:

"Error" property

Error code of an error

```
int32 Error { get; }
```

"Name" property

Name of the associated object of the data source

```
string Name { get; }
```

Example

Error output when writing a TagSet:

Copy code

```
public void WritePartlyNotExistingTagSetSync()
{
   ITagSet odkTagSet = runtime.GetObject<ITagSet>();
   odkTagSet.Add("Tag1", 1);
   odkTagSet.Add("Tag2", 2);
   odkTagSet.Add("NotExistingTag1", 1);
   odkTagSet.Add("NotExistingTag2", 2);

  IList<IErrorResult> writeResult = odkTagSet.Write();

   foreach (var result in writeResult)
   {
      System.Console.WriteLine(string.Format("Write tag '{0}' failed, error code {1}",
result.Name, result.Error));
   }
}
```

See also

IRuntime (Page 7834)

IErrorInfo (Page 7842)

IErrorInfo

Description

The C# interface "IErrorInfo" specifies methods and properties for handling error codes.

Members

The following properties and methods are specified in the interface:

"Error" property

Error code of an error

```
int32 Error { get; }
```

"GetErrorDescription" method

Output an error description for the error code.

```
string GetErrorDescription(uint32 Error)
```

`Error`
Error code that is passed by the ODK client.

See also

IErrorResult (Page 7841)

OdkException

Description

In the case of exceptions, the ODK triggers an OdkException in the .Net environment. The OdkException can be caught by try-catch blocks and evaluated.

The "OdkException" class inherits all properties and methods of the .NET class "Exception".

Members "OdkException"

The following objects and methods are also implemented in the "OdkException" class for all properties and methods of the .NET class "Exception".

"OdkException" method

- Trigger exception without alarm.
  `OdkException()`
- Trigger exception with alarm and error description.
  `OdkException(string message)`
  `message`
  Description of the error
- Trigger exception with alarm and error description. Trigger additional exception with reference to triggering exception.
  `OdkException(string message, Exception innerException)`
  - `message`
    Description of the error
  - `innerException`
    Triggering exception
- Trigger exception with serialized data.
  `OdkException(SerializationInfo info, StreamingContext context)`
  - `info`
    Serialized data of the exception
  - `context`
    Describes the origin or target of the serialized data.

"ErrorCode" property

Error code of the exception

```
UInt32 ErrorCode { get; }
```

"ErrorSubCategory" property

Subcategory of an error code

```
UInt32 ErrorSubCategory { get; }
```

Example

Exception handling using the example of optional components:

Copy code

```
public void GetOptionObject()
{
    //load option component by name
    IMyOption rtOption = (IMyOption)runtime.GetOption("MyOptionName");
    //create a instance of the option object IMyOptionObject
    IMyOptionObject optionObject = rtOption.GetObject<IMyOptionObject>();
    try
    {
        string strMethod = optionObject.MyMethod();
        string strProperty = optionObject.MyProperty;
    }
    catch (OdkException ex)
    {
        //It is an option error?
        if (ex.ErrorSubCategory == MyOptionConstants.MYOPTION_ERRORCATEGORY)
        {
            //Handle option specific error
            if (ex.ErrorCode == (uint)MyErrorCodes.E_UNKNOWN_NAME)
            {
                //get error description
                string errorDescription = ex.Message;
            }
        }
    }
}
```

20.3.8.4 Interfaces of the tags

IProcessValue

Description

The C# interface "IProcessValue" specifies properties and methods for values of process tags of the Runtime system. The "IProcessValue" interface provides values from the result of a read operation or monitoring.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; }
```

"Value" property

Value of the tag at the moment of the read operation.

```
object Value { get; }
```

"Quality" property

Quality code of the read operation of the tag.

```
uint32 Quality { get; }
```

"TimeStamp" property

Time stamp of the last successful read operation of the tag.

```
DateTime TimeStamp { get; }
```

"Error" property

Error code of the last read or write operation of the tag.

```
int32 Error { get; }
```

Example

Output properties of the "IProcessValue" object that is returned by the ITag.Read method:

Copy code

```
public void ReadSingleTagSync()
{
   var myTag = runtime.GetObject<ITag>("Tag1");
   var value = myTag.Read(HmiReadType.Cache); // Reads value from Cache
   Console.WriteLine("Name: {0} Timestamp: {1} Value: {2} Quality: {3}", value.Name, value.
TimeStamp, value.Value, value.Quality);
}
```

See also

ITag (Page 7845)

ITagSet (Page 7848)

ITagSetQCD (Page 7855)

ITag

Description

The C# interface "ITag" specifies properties and methods for handling tags of the Runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the tag that is read with the "Read" method.

```
string Name { get; set; }
```

"Read" method

Read process value and properties of the tag synchronously from the Runtime system.

```
IProcessValue Read(HmiReadType type = HmiReadType.Cache)
```

`type`
The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default parameter): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag value directly from the AS. The tag image is not used.

"Write" method

Write process value of the tag synchronously in the Runtime system.

```
void Write(
     object value,
     HmiWriteType type = HmiWriteType.NoWait)
```

- `value`
  Value of the tag
- `type`
  The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType.NoWait` (default parameter): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType.Wait`: Waits until the tag value is written in the AS. If an error occurs during the write operation, an exception is triggered.

"WriteQCD" method

Write process value with quality code of the tag synchronously in the Runtime system. The tag also has a freely definable time stamp. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

```
void Write(
     object value,
     DateTime timeStamp,
     uint32 qualityCode,
     HmiWriteType type = HmiWriteType.NoWait)
```

- `value`
  Value of the tag
- `timeStamp`
  Time stamp of the tag
- `qualityCode`
  Quality code of the tag
- `type`
  The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType.NoWait` (default parameter): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType.Wait`: Waits until the tag value is written in the AS. If an error occurs during the write operation, an exception is triggered.

"WriteWithOperatorMessage" method

Write process value of the tag synchronously in the runtime system and create operator input alarm. In addition to the reason, the operator input alarm contains the old and new value, the user and host names and the unit.

```
void WriteWithOperatorMessage(
     object value,
     string reason)
```

- `value`
  Value of the tag
- `reason`
  Reason for the value change for alarm

Example

Read and write tag synchronously:

```
public void ReadSingleTagSync()
{
   var myTag = runtime.GetObject<ITag>("Tag1");
   var value = myTag.Read(HmiReadType.Cache); // Reads value from Cache
   Console.WriteLine("Name: {0} Timestamp: {1} Value: {2} Quality: {3}", value.Name, value.
TimeStamp,value.Value, value.Quality);
}

public void WriteSingleTagSync()
{
   var odkTag = runtime.GetObject<ITag>("Tag1");
   var value = 5;
   odkTag.Write(value, HmiWriteType.NoWait);   // Writes value without waiting that value h
as been written to PLC
   var pvalue = odkTag.Read(HmiReadType.Cache);
}
```

See also

IProcessValue (Page 7844)

ITagSet (Page 7848)

ITagSet

Description

The C# interface "ITagSet" specifies properties, methods and events for optimized access to several tags of the Runtime system.

After initialization of the "ITagSet" object, you can execute read and write access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Note

You have the option to enumerate via a TagSet and to access a special element via the name. This is useful if you want to change the value of a tag in the TagSet for a write operation. To write the values in the process image, a "Write" or "WriteAsync" method must first be called:

```
MyTagSet.Add("MyTag1", "MyTag2");
MyTagSet["MyTag1"] = 5; // Set value to 5 for write operation
MyTagSet.Write();
```

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a TagSet. If several TagSets are used to read tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of tags of a TagSet list

```
int32 Count { get; }
```

"Add" method

Add tag to a TagSet.

Add tag with or without process value to the TagSet:

```
void Add(ICollection<string> tagNames)
```

`tagNames`
List with tag names for TagSet

or

```
void Add(string tagName, object value = null)
```

- `TagName`
  Name of the tag for TagSet
- `value`
  New value of the tag, default: No value

"Remove" method

Remove individual tag from a TagSet.

```
void Remove(string tagName)
```

`tagName`
Name of the tag that is removed from TagSet.

"Clear" method

Remove all tags from a TagSet.

```
void Clear()
```

"Read" method

Read process values and properties of all the tags of a TagSet synchronously from the Runtime system.

```
IList<IProcessValue> Read(HmiReadType type = HmiReadType.Cache)
```

`type`
The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag values directly from the automation system. The tag image is not used.

"ReadAsync" method

Read process values and properties of all the tags of a TagSet asynchronously from the Runtime system.

```
void ReadAsync(HmiReadType type = HmiReadType.Cache)
```

`type`
The enumeration "HmiReadType" specifies the origin of the tag value:

- `HmiReadType.Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
- `HmiReadType.Device`: Reads the tag values directly from the automation system. The tag image is not used.

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
IList<IErrorResult> Write(HmiWriteType type = HmiWriteType.NoWait)
```

`type`
The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:

- `HmiWriteType.NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
- `HmiWriteType.Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType.Wait` type and waits until the tag value has been written in the automation system. If an error occurs during the write operation, it is reported via the AsyncHandler.

```
void WriteAsync()
```

"WriteWithOperatorMessage" method

Write process values of all tags of a TagSet synchronously in the Runtime system and create operator input alarms. In addition to the reason, the operator input alarms contain the old and new value, the user and host names and the unit.

```
void WriteWithOperatorMessage(string reason)
```

`reason`
Reason for the value change for alarm

"Subscribe" method

Subscribe all tags of a TagSet asynchronously for cyclic monitoring of the process values.

Note

Tags from IO devices with the "Cyclic in operation" acquisition mode

For a tag with the acquisition mode "Cyclic in operation", the value stored in the process image when Subscribe is called might be outdated. OnAdd therefore only provides the QualityCode "uncertain". Only value changes made after the Subscribe call provide the current value and the QualityCode "good".

```
void Subscribe()
```

"CancelSubscribe" method

Cancel monitoring of all tags of a TagSet.

```
void CancelSubscribe()
```

"OnReadResult" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnReadResultHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnReadResultHandler OnReadResult
```

"OnWriteResult" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnWriteResultHandler" delegate.

Declares the event and the event handler for asynchronous write operations.

```
event OnWriteResultHandler OnWriteResult
```

"OnDataChanged" event

After a process value change of a monitored TagSet, the event calls an instance of the "OnDataChangedHandler" delegate.

Declares the event and the event handler for process value changes.

```
event OnDataChangedHandler OnDataChanged
```

"OnReadResultHandler" delegate

Specifies the signature of the event handling method for the "OnReadResult" event of a TagSet.

```
void OnReadResultHandler(
        ITagSet sender,
        IList<IProcessValue> values,
        bool completed)
```

- `sender`
  The read out "TagSet" object
- `values`
  Event data as a list of "IProcessValue" objects of the read tag
- `completed`
  Status of the asynchronous transfer:
  - True: All values of the TagSet are read.
  - False: Not all values of the TagSet are read yet.

"OnWriteResultHandler" delegate

Specifies the signature of the event handling method for the "OnWriteResult" event of a TagSet.

```
void OnWriteResultHandler(
        ITagSet sender,
        IList<IErrorResult> values,
        bool completed)
```

- `sender`
  The written "TagSet" object
- `values`
  Error during write operations of tags as "IErrorResult" object
- `completed`
  Status of the asynchronous transfer:
  - True: All values of the TagSet are written.
  - False: Not all values of the TagSet are written yet.

"OnDataChangedHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event of a TagSet.

```
void OnDataChangedHandler(
        ITagSet sender,
        IList<IProcessValue> values)
```

- `sender`
  The monitored "TagSet" object
- `values`
  Event data as a list of "IProcessValue" objects of the changed process values

Example

Read TagSet synchronously and write with change:

Copy code

```
public void ReadTagSetSync()
{
   var odkTagSet = runtime.GetObject<ITagSet>())
   odkTagSet.Add(new[] { "Tag1", "Tag2" });
   var values = odkTagSet.Read();
   foreach (var value in values)
   {
      Console.WriteLine("Name: {0} Timestamp: {1} Value: {2} Quality: {3}", value.Name,
value.TimeStamp, value.Value, value.Quality);
      // Get Type information
      PrintProcessValue(value);
   }
}


public void WriteTagSetSyncWithChange()
{
   var odkTagSet = runtime.GetObject<ITagSet>();
   odkTagSet.Add(new[] { "Tag1", "Tag2" });
   // Modify the value of a tag in the tagset and write
    odkTagSet["Tag1"] = 5;
    odkTagSet.Write();
}
```

Define functions for asynchronous write operations and define the monitoring of TagSets.

Monitor events "OnWriteResult" or "OnDataChanged" and call the event handling methods "odkTagSet_OnWriteComplete" or "odkTag_OnDataChanged" on occurrence:

Copy code

```
public void WriteTagSetAsync()
{
   var odkTagSet = runtime.GetObject<ITagSet>();
   odkTagSet.Add("Tag1", 1);
   odkTagSet.Add("Tag2", 2);
   // Assign callback function
   odkTagSet.OnWriteResult += OdkTagSet_OnWriteResult;
   odkTagSet.WriteAsync();
   _event.WaitOne();
   _event.Reset();
}

private void OdkTagSet_OnWriteResult(ITagSet sender, IList<IErrorResult> values, bool compl
eted)
{
   foreach (var value in values)
   {
      Console.WriteLine("Error = {0} Name = {1}", value.Error, value.Name);
   }
}

public void SubscribeTagSet()
{
   var odkTagSet = runtime.GetObject<ITagSet>();
   odkTagSet.Add(new[] { "Tag1", "Tag2" });
   // Assign callback function
   odkTagSet.OnDataChanged += OdkTagSet_OnDataChanged;
   odkTagSet.Subscribe();
   _event.WaitOne();
   _event.Reset();
   odkTagSet.CancelSubscribe();
}

public void OdkTagSet_OnDataChanged(ITagSet sender, IList<IProcessValue> pItems)
{
   foreach (var value in pItems)
      Console.WriteLine("Name: {0} Timestamp: {1} Value: {2} Quality: {3}", value.Name, val
ue.TimeStamp, value.Value, value.Quality);
}
```

See also

IProcessValue (Page 7844)

ITag (Page 7845)

ITagSetQCD

Description

The C# interface "ITagSetQCD" specifies properties, methods and events for optimized writing of several tags of the Runtime system. The tags also have a freely definable time stamp and quality code. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

After initialization of the "ITagSetQCD" object, you can have read access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Note

You have the option to enumerate via a TagSet and to access a special element via the name. This is useful if you want to change the value of a tag in the TagSet for a write operation. To write the values in the process image, a "Write" or "WriteAsync" method must first be called:

```
MyTagSet.Add("MyTag1", "MyTag2");
MyTagSet["MyTag1"] = 5; // Set value to 5 for write operation
MyTagSet.Write();
```

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identification characteristics of a TagSet. If several TagSets are used to read tags, you can assign the response to the request via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of tags of a TagSet list

```
int32 Count { get; }
```

"Add" method

Add user-defined tag with process value, quality code and time stamp to the TagSet.

```
void Add(string tagName, object value, DateTime timeStamp, uint32
qualityCode)
```

- `TagName`
  Name of the tag for TagSet
- `value`
  New value of the tag
- `timeStamp`
  Time stamp of the tag
- `qualityCode`
  Quality code of the tag

"Remove" method

Remove individual tag from a TagSet.

```
void Remove(string tagName)
```

`tagName`
Name of the tag that is removed from TagSet.

"Clear" method

Remove all tags from a TagSet.

```
void Clear()
```

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
IList<IErrorResult> Write(HmiWriteType type = HmiWriteType.NoWait)
```

`type`
The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:

- `HmiWriteType.NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
- `HmiWriteType.Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType.Wait` type and waits until the tag value has been written in the automation system. If an error occurs during the write operation, it is reported via the AsyncHandler.

```
void WriteAsync()
```

"OnWriteComplete" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnWriteCompleteTagSetQCDHandler" delegate.

Declares the event and the event handler for asynchronous write operations with quality codes and time stamps.

```
event OnWriteCompleteTagSetQCDHandler OnWriteComplete
```

"OnWriteCompleteTagSetQCDHandler" delegate

Specifies the signature of the event handling method for the "OnWriteComplete" event of a TagSet with quality code and time stamps.

```
void OnWriteCompleteTagSetQCDHandler(
        ITagSetQCD sender,
        IList<IErrorResult> pItems)
```

- `sender`
  Source of the event
- `pItems`
  Error during write operations of tag as "IErrorResult" object

Example

Write TagSet with time stamp and quality code synchronously:

Copy code

```
public void WriteTagSetQCDSync()
{
    var odkTagSet = runtime.GetObject<ITagSetQCD>();
    odkTagSet.Add("Tag1", 1, DateTime.Now, 128);
    odkTagSet.Add("Tag2", 2, DateTime.Now, 128);
    var writeResult = odkTagSet.Write();
}
```

See also

IProcessValue (Page 7844)

ITagSetQCDItem (Page 7858)

ITagSetQCDItem

Description

The C# interface "ITagSetQCDItem" specifies properties for user-defined values of tags in a TagSetQCD. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, only the tag value is set. The QualityCode and time stamp are set internally by the system.

The objects "ITagSetQCDItem" are not used by the "ITagSetQCD" object and can also be edited with the methods of the "ITagSetQCD" interface.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; set; }
```

"Value" property

Tag value

```
object Value { get; set; }
```

"Quality" property

Quality code of the tag

```
int32 Quality { get; set; }
```

"TimeStamp" property

Time stamp of the tag

```
DateTime TimeStamp { get; set; }
```

See also

ITagSetQCD (Page 7855)

ILoggedTagValue

Description

The C# interface "ILoggedTagValue" specifies the properties that a logged process value of a logging tag has in the logging system.

An "ILoggedTagValue" instance is a pure data object. The instance encapsulates all properties of the logged process value. It represents a historical process value.

Members

The following properties are specified in the interface:

"Name" property

Name of the logging tag

```
string Name { get; set; }
```

"Value" property

The logged process value

```
object Value { get; set; }
```

"Quality" property

Quality code of the process value

```
int16 Quality { get; set; }
```

If the "ILoggedTagValue" instance has been added to the log by calling the "Write" method, "Quality" has the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

"TimeStamp" property

Time stamp of the process value

```
DateTime TimeStamp { get; set; }
```

"Error" property

Error code of the process value

```
uint32 Error { get; }
```

"Flags" property

Context information from the read operation for the process value

```
HmiTagLoggingValueFlags Flags { get; set; }
```

The "HmiTagLoggingValueFlags" enumeration contains the following bit-by-bit-coded values:

- 0: `Extra`
  There are still additional values at the time of the process value.
- 2: `Calculated`
  Process value is calculated.

- 16: `Bounding`
  Process value is a limit value.
- 32: `NoData`
  No additional information available
- 64: `FirstStored`
  Process value is the first value stored in the logging system.
- 128: `LastStored`
  Process value is the last value stored in the logging system.

See also

ILoggedTag (Page 7860)

ILoggedTagSet (Page 7861)

ILoggedTag

Description

The C# interface "ILoggedTag" specifies properties and methods for the handling of logging tags of a logging system. A logging tag is represented by an "ILoggedTag" instance. The information on the logged process values of the logging tag is stored in "ILoggedTagValue" instances.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the logging tag

```
string Name { get; set; }
```

"Read" method

Synchronous read operation.

The method returns a list of the "ILoggedTagValue" instances of the logging tag whose time stamps are in the time period defined by the arguments.

```
IList<ILoggedTagValue> Read(DateTime begin, DateTime end, bool
boundingValue)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `boundingValue`
  True, to additionally return high and low limits.

"Write" method

Synchronous read operation.

The method manually writes an "ILoggedTagValue" instance to the logging system. "Quality" of the instance receives the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

Application example: While the connection to the logging system is interrupted, the process value of a tag that is being logged changes. The resulting "ILoggedTagValue" instance is cached. After the connection has been re-established, manually add the instance to the log by calling "Write".

```
void Write(ILoggedTagValue tag)
```

- `tag`
  The "ILoggedTagValue" instance to be logged

Example

Output process values of the logging tag "Tag1:Tag1Logging1" for a time period:

Copy code

```
public void ReadSingleTag()
{
   var tag = _runtime.GetObject<ILoggedTag>("Tag1:Tag1Logging1";
   var begin = DateTime.UtcNow.AddMinutes(-10);
   var end = DateTime.UtcNow;
   var values = tag.Read(begin, end, true);
   foreach (var v in values)
   {
      var pinfo = _runtime.GetObject<IErrorInfo>();
      Console.WriteLine();
      Console.WriteLine("Name: {0} Error:{1}", v.Name, pinfo.GetErrorDescription(error));
   }
}
```

See also

ILoggedTagValue (Page 7859)

ILoggedTagSet (Page 7861)

ILoggedTagSet

Description

The C# interface "ILoggedTagSet" specifies properties, methods and events for optimized access to a collection of "ILoggedTag" instances of a logging system.

After initializing an "ILoggedTagSet" instance, you can read or write multiple "ILoggedTag" instances in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple logging tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

Identifier of an "ILoggedTagSet" instance. If several "ILoggedTagSet" instances are used to read logging tags, you can assign the response to the job via ContextId.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

"Count" property

Number of "ILoggedTag" instances of the "ILoggedTagSet" instance

```
int32 Count { get; }
```

"Add" method

The method is overloaded:

- Add multiple "ILoggedTag" instances to the "ILoggedTagSet" instance:
  `void Add(ICollection<string> tagNames)`
  `tagNames`
  A collection with the names of the "ILoggedTag" instances
- Add a single "ILoggedTag" instance to the "ILoggedTagSet" instance
  `void Add(string tagName)`
  `tagName`
  Name of the "ILoggedTag" instance
- Add an "ILoggedTagValue" instance to the "ILoggedTagSet" instance:
  `void Add(ILoggedTagValue object)`
  `object`
  Reference of the "ILoggedTagValue" instance

"Remove" method

Remove individual logging tag from the "ILoggedTagSet" instance.

```
void Remove(string tagName)
```

`tagName`
Name of the logging tag that is removed.

"Clear" method

Remove all logging tags from an "ILoggedTagSet" instance.

```
void Clear()
```

"Read" method

Synchronous read operation.

Reads from all logging tags of the "ILoggedTagSet" instance the logged process values whose time stamp is in the time period defined by the arguments.

```
IList<ILoggedTagValue> Read(DateTime begin, DateTime end, bool
boundingValue)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `boundingValue`
  True, to additionally return high and low limits.

"ReadAsync" method

Asynchronous read operation.

Reads from all logging tags of the "ILoggedTagSet" instance the logged process values whose time stamp is in the time period defined by the arguments.

```
void ReadAsync(DateTime begin, DateTime end, bool boundingValue)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `boundingValue`
  True, to additionally return high and low limits.

"Subscribe" method

Subscribe all logging tags of an "ILoggedTagSet" instance asynchronously to update the process values when they change. When new process values are logged, they can be processed with the "OnDataChanged" event.

```
void Subscribe()
```

"CancelSubscribe" method

Revoke update of process values on change for all logging tags of an "ILoggedTagSet" instance.

```
void CancelSubscribe()
```

"Write" method

Synchronous write operation.

Writes the "ILoggingTagValue" instances added by call of "Add" to the logging system. "Quality" of the instance receives the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

Errors during write operations are returned in a list with "IErrorResult" instances.

Application example: The connection to the archive server was interrupted. After the connection has been re-established, add multiple "ILoggedTagValue" instances cached in the PLC to the archive afterwards by calling "Write".

```
IList<IErrorResult> Write()
```

"WriteAsync" method

Asynchronous write operation.

Writes the "ILoggingTagValue" instances added by call of "Add" to the logging system. "Quality" of the instance receives the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

Application example: The connection to the archive server was interrupted. After the connection has been re-established, add multiple "ILoggedTagValue" instances cached in the PLC to the archive afterwards by calling "Write".

```
void WriteAsync()
```

"OnReadComplete" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnReadCompleteTagSetHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnReadCompleteTagSetHandler OnReadComplete
```

"OnDataChanged" event

After a process value change of a monitored "ILoggedTagSet" instance, the event calls an instance of the "OnDataChangedTagSetHandler" delegate.

Declares the event and the event handler for process value changes.

```
event OnDataChangedTagSetHandler OnDataChanged
```

"OnReadCompleteTagSetHandler" delegate

Specifies the signature of the event handling method for the "OnReadComplete" event of an "ILoggedTagSet" instance.

```
void OnReadCompleteTagSetHandler(
        ILoggedTagSet sender,
        uint32 errorCode,
        IList<ILoggedTagValue> values,
        bool completed)
```

- `sender`
  Source of the event
- `errorCode`
  Error during asynchronous transfer
- `values`
  Event data as a list of "ILoggedTagValue" instances of the read logging tags
- `completed`
  Status of the asynchronous transfer:
  - True: All values of the "ILoggedTagSet" instance are read out.
  - False: All values of the "ILoggedTagSet" instance are not yet read out.

"OnDataChangedTagSetHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event of an "ILoggedTagSet" instance.

```
void OnDataChangedTagSetHandler(
        ILoggedTagSet sender,
        uint32 errorCode,
        IList<ILoggedTagValue> value)
```

- `sender`
  Source of the event
- `errorCode`
  Error during asynchronous transfer
- `value`
  Event data as a list of "ILoggedTagValue" instances with process values of the changed logging tags

Example

Read and output process values of logging tags of a specified period in the "ILoggedTagSet" instance "tagSet" asynchronously:

Copy code

```
public void ReadTagSetAsync()
{
    try
    {
        var begin = DateTime.UtcNow.AddHours(-1);
        var end = DateTime.UtcNow;
        var tagSet = _runtime.GetObject<ILoggedTagSet>();
        tagSet.OnReadComplete += TagSet_OnReadComplete;
        tagSet.Add("Tag1:Tag1Logging1");
        tagSet.Add("Tag2:Tag2Logging1");
        tagSet.ReadAsync(begin, end, true);
    }
    catch (OdkException ex)
    {
        Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

private void TagSet_OnReadComplete(ILoggedTagSet sender, uint errorCode,
IList<ILoggedTagValue> values, bool completed)
{
    Print(values);
    sender.Dispose();
}
```

Output process values of logging tag "Tag1:Tag1Logging1" in case of change:

Copy code

```
public void SubscribeTagSet()
{
    try
    {
        ILoggedTagSet tagSet = _runtime.GetObject<ILoggedTagSet>();
        tagSet.Add("Tag1:Tag1Logging1");
        tagSet.OnDataChanged += tagSet_OnDataChanged;
        tagSet.Subscribe();
    }
    catch (OdkException ex)
    {
        Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

void tagSet_OnDataChanged(ILoggedTagSet sender, UInt32 errorCode, IList<ILoggedTagValue>
values)
{
    Print(values);
    sender.Dispose();
}
```

See also

ILoggedTag (Page 7860)

ILoggedTagValue (Page 7859)

ITags

Description

The C# interface "ITags" defines methods with which you can access configured tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

"Find" method

Supplies a collection with "ITagAttributes" instances of the specified tag.

```
ICollection<ITagAttributes>  Find(ICollection<string> SystemNames,
string Filter = null, UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcards (*).
- (Optional) `LanguageID`
  Language code ID of filter

"FindAsync" method

For asynchronous searching for "ITagAttributes" instances of the specified tags.

```
void FindAsync(IList<string> SystemNames, string Filter = null,
UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcards (*)
- (Optional) `LanguageID`
  Language code ID of filter

"OnTagAttributesRecieved" event

After completion of the "FindAsync" method, the event calls an instance of the "OnTagAttributesRead" delegate.

```
event OnTagAttributesRead OnTagAttributesRecieved;
```

"OnTagAttributesRead" delegate

Specifies the signature of the event handling method for the "OnTagAttributesReceived" event of an "ITagAttributes" instance.

```
public delegate void OnTagAttributesRead(ITags sender,
ICollection<ITagAttributes> tagAttributes, bool completed)
```

- `sender`
  Source of the event
- `tagAttributes`
  Event data as collection of "ITagAttributes" instances
- `completed`
  Status of the asynchronous transfer:
  - True: All tags have been read out.
  - False: Not all tags have been read out yet.

ITagAttributes

Description

The C# interface "ITagAttributes" defines properties of a tag.

Member

"Name" property

The name of the tag. Must be unique throughout the device.

```
string Name { get; }
```

"DisplayName" property

The display name of the tags

```
string DisplayName { get; }
```

"DataType" property

The data type of the tag

```
HmiDataType DataType { get; }
```

"Connection" property

The connection of the tag

The memory location of the tag in the controller is accessed via the connection.

```
string Connection { get; }
```

"AcquisitionCycle" property

The acquisition cycle of tags

If you configure a tag at an object, the acquisition cycle of the tag is displayed at the object.

```
string AcquisitionCycle { get; }
```

"AcquisitionMode" property

The acquisition mode of the tag

```
HmiAcquisitionMode AcquisitionMode { get; }
```

The enumeration "HmiAcquisitionMode" can contain the following values:

- Undefined (0)
- CyclicOnUse (1)
- CyclicContinous (2)
- OnDemand (3)
- OnChange (4)

"MaxLength" property

The length of the tags.

The length is only available with a string tag. The length is preset for structure tags and cannot be changed.

```
UInt32 MaxLength { get; }
```

"Persistent" property

The persistence of the tags

```
bool Persistent { get; }
```

"InitialValue" property

The start value of the tag

After Runtime starts, the tag retains the start value until an operator or the PLC changes the value.

```
object InitialValue { get; }
```

"InitialMaxValue" property

The start value for the event "On exceeding".

```
object InitialMaxValue { get; }
```

"InitialMinValue" property

The start value for the event "On falling below"

```
object InitialMinValue { get; }
```

"SubstituteValue" property

The substitute value of the tag

If the selected condition occurs, the tag is filled with the substitute value in runtime.

```
object SubstituteValue { get; }
```

"SubstituteValueUsage" property

The condition for using the substitute value of the tag

```
HmiSubstituteValueUsage SubstituteValueUsage { get; }
```

The enumeration "HmiSubstituteValueUsage" can contain the following values:

- None (0)
- InvalidValue (1)
- RangeViolation (2)
- InvalidValueOrRangeViolation (3)

ILoggingTags

Description

The C# interface "ILoggingTags" defines methods with which you can access configured logging tags.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

Members

"Find" method

Supplies a collection with "ITagAttributes" instances of the specified logging tag.

```
ICollection<ILoggingTagAttributes>  Find(IList<string> SystemNames,
string Filter = null, UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*).
  Example:
  `Tag1:*` Supplies all logging tags of "Tag1".
- (Optional) `LanguageID`
  Language code ID of filter

"FindAsync" method

For asynchronous searching for "ILoggingTagAttributes" instances.

```
void FindAsync(IList<string> SystemNames, string Filter = null,
UInt32 LanguageID = 0)
```

- `SystemNames`
  Collection of SystemNames on which the tags were configured.
- (Optional) `Filter`
  Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*).
  Example:
  `Tag1.*`: Supplies all logging tags of "Tag1".
- (Optional) `LanguageID`
  Language code ID of filter

"OnLoggingTagAttributesReadReceived" event

After completion of the "FindAsync" method, the event calls an instance of the "OnLoggingTagAttributesRead" delegate.

```
event OnLoggingTagAttributesRead OnLoggingTagAttributesReadRecieved;
```

"OnLoggingTagAttributesRead" delegate

Specifies the signature of the event handling method for the "OnLoggingTagAttributesRead" event of an "ILoggingTags" instance.

```
public delegate void OnLoggingTagAttributesRead(ILoggingTags
sender, ICollection<ILoggingTagAttributes> tagAttributes, bool
completed);
```

- `sender`
  Source of the event
- `tagAttributes`
  Event data as collection of "ILoggingTagAttributes" instances
- `completed`
  Status of the asynchronous transfer:
  - True: All logging tags are read out.
  - False: Not all logging tags are read out yet.

ILoggingTagAttributes

Description

The C# interface "ILoggingTagAttributes" defines properties of a logging tag.

Member

"Name" property

The name of the tag

```
string Name { get; }
```

"DisplayName" property

The display name of the tags

```
string DisplayName { get; }
```

"DataType" property

The data type of the tag

```
HmiDataType DataType { get; }
```

20.3.8.5 Interfaces of the alarms

IAlarmResult

Description

The C# interface "IAlarmResult" specifies properties for accessing properties of active alarms of the Runtime system.

An "IAlarmResult" object is a pure data object which maps all properties of an active alarm.

Members

The following properties are specified in the interface:

"InstanceID" property

InstanceID for an alarm with multiple instances

```
uint32 InstanceID { get; }
```

"SourceID" property

Source at which the alarm was triggered.

```
string SourceID { get; }
```

"Name" property

Name of the alarm

```
string Name { get; }
```

"AlarmClassName" property

Name of the alarm class

```
string AlarmClassName { get; }
```

"AlarmClassSymbol" property

Symbol of the alarm class

```
string AlarmClassSymbol { get; }
```

"AlarmParameterValues" property

Parameter values of the alarm

```
IReadOnlyList<object> AlarmParameterValues { get; }
```

"AlarmText1" ... "AlarmText9" properties

Additional texts 1-9 of the alarm

```
string AlarmText1 { get; }
```

...

```
string AlarmText9 { get; }
```

"ChangeReason" property

Trigger event for modification of the alarm state

```
uint16 ChangeReason { get; }
```

"Connection" property

Connection via which the alarm was triggered

```
string Connection { get; }
```

"State" property

Current alarm state

```
HmiAlarmState State { get; }
```

The enumeration "HmiAlarmState" can contain the following values:

- Standard (0x00)
- Raised (0x01)
- RaisedCleared (0x02)
- RaisedAcknowledged (0x05)
- RaisedAcknowledgedCleared (0x06)
- RaisedClearedAcknowledged (0x07)
- Removed (0x80)

"StateText" property

Current alarm state as text, for example, "active" or "inactive"

```
string StateText { get; }
```

"EventText" property

Text that describes the alarm event.

```
string EventText { get; }
```

"InfoText" property

Text that describes an operator instruction for the alarm.

```
string InfoText { get; }
```

"TextColor" property

Text color of the alarm state

```
uint32 TextColor { get; }
```

"BackColor" property

Background color of the alarm state

```
uint32 BackColor { get; }
```

"Flashing" property

Indicates whether the alarm is flashing.

```
bool Flashing { get; }
```

"InvalidFlags" property

Identification of the alarm with invalid data

```
byte InvalidFlags { get; }
```

"ModificationTime" property

Time of the last modification to the alarm state

```
DateTime ModificationTime { get; }
```

"RaiseTime" property

Time the alarm was triggered

```
DateTime RaiseTime { get; }
```

"AcknowledgementTime" property

Time of alarm acknowledgment

```
DateTime AcknowledgementTime { get; }
```

"ClearTime" property

Time of alarm reset

```
DateTime ClearTime { get; }
```

"ResetTime" property

Time of alarm reset

```
DateTime ResetTime { get; }
```

"SuppressionState" property

Status of alarm visibility

```
byte SuppressionState { get; }
```

"SystemSeverity" property

Severity of the system error

```
UInt8 SystemSeverity { get; }
```

"Priority" property

Relevance for display and sorting of the alarm

```
UInt8 Priority { get; }
```

"Origin" property

Origin for display and sorting of the alarm

```
string Origin { get; }
```

"Area" property

Origin area for display and sorting of the alarm

```
string Area { get; }
```

"Value" property

Current process value of the alarm

```
object Value { get; }
```

"ValueQuality" property

Quality of the process value of the alarm

```
uint16 ValueQuality { get; }
```

"ValueLimit" property

Process value limit of the alarm

```
object ValueLimit { get; }
```

"UserName" property

User name of the operator input alarm

```
string UserName { get; }
```

"UserResponse" property

Expected or required user response to the alarm

```
byte UserResponse { get; }
```

"HostName" property

Name of the host that triggered the alarm.

```
string HostName { get; }
```

"Id" property

ID of the alarm that is also used in the display.

```
Uint32 Id { get; }
```

"AlarmGroupID" property

ID of the alarm group to which the alarm belongs.

```
UInt32 AlarmGroupID { get; }
```

"SourceType" property

Source from which the alarm was generated, for example, tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

The enumeration "HmiAlarmSourceType" can contain the following values:

- `Undefined` (0)
- `Tag` (1)
- `Controller` (2)
- `System` (3)
- `Alarm` (4)

"DeadBand" property

Range of the triggering tag in which no alarms are generated.

```
object DeadBand { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm control to its origin.

```
string LoopInAlarm { get; }
```

"NotificationReason" property

Reason for the notification

```
HmiNotificationReason NotificationReason { get; }
```

The enumeration "HmiNotificationReason" can contain the following values:

- `Unknown` (0)
- `Add` (1)
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- `Modify` (2)
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

Note

Changes to the alarms only lead to notifications again when the alarm meets the filter criteria again. In this case, "NotificationReason" is set to `Add`.

Note

Removing alarm from business logic

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- State-based monitoring: The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.

Example:

An ODK client begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the ODK client of the "NotificationReason" as follows:

Notification-Reason	Description
Add	• The "State" property is 1. The alarm is active.
Modify	• The "State" property has not changed. • Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, for example, alarm is inactive.

"Duration" property

Returns the time interval in nanoseconds between triggering of the alarm and its previous status change.

```
TimeSpan  Duration { get; }
```

Example

When alarms become active, a selection of properties of the "IAlarmResult" objects is output:

Copy code

```
public void alarm_OnAlarmHandler(IAlarmSubscription sender, UInt32 nGlobalError, String
systemName, IList<IAlarmResult> value, bool completed)
{
    if (0 == nGlobalError)
    {
        try
        {
            AlarmList = new List<IALarmResult>();
            foreach (var item in value)
            {
               System.Console.WriteLine(string.Format("InstanceID: {0}", item.InstanceID));
                System.Console.WriteLine(string.Format("AcknowledgementTime: {0}",
item.AcknowledgementTime));
                System.Console.WriteLine(string.Format("AlarmClass: {0}",
item.AlarmClassName));
                System.Console.WriteLine(string.Format("AlarmClassSymbol: {0}",
item.AlarmClassSymbol));
                System.Console.WriteLine(string.Format("Id: {0}", item.Id));
                System.Console.WriteLine(string.Format("AlarmParameterValues: {0}",
item.AlarmParameterValues));
              System.Console.WriteLine(string.Format("AlarmText1: {0}", item.AlarmText1));
              System.Console.WriteLine(string.Format("AlarmText9: {0}", item.AlarmText9));
                System.Console.WriteLine(string.Format("Area: {0}", item.Area));
                System.Console.WriteLine(string.Format("BackColor: {0}", item.BackColor));
                System.Console.WriteLine(string.Format("ChangeReason: {0}",
item.ChangeReason));
                System.Console.WriteLine(string.Format("ClearTime: {0}", item.ClearTime));
              System.Console.WriteLine(string.Format("Connection: {0}", item.Connection));
                System.Console.WriteLine(string.Format("DeadBand: {0}", item.DeadBand));
                System.Console.WriteLine ("Duration: {0}", item.Duration);
                System.Console.WriteLine(string.Format("EventText: {0}", item.EventText));
                System.Console.WriteLine(string.Format("InfoText: {0}", item.InfoText));
                System.Console.WriteLine(string.Format("Flashing: {0}", item.Flashing));
                System.Console.WriteLine(string.Format("HostName: {0}", item.HostName));
                System.Console.WriteLine(string.Format("InvalidFlags: {0}",
item.InvalidFlags));
                System.Console.WriteLine(string.Format("LoopInAlarm: {0}",
item.LoopInAlarm));
                System.Console.WriteLine(string.Format("ModificationTime: {0}",
item.ModificationTime));
                System.Console.WriteLine(string.Format("Name: {0}", item.Name));
                System.Console.WriteLine(string.Format("Origin: {0}", item.Origin));
                System.Console.WriteLine(string.Format("Priority: {0}", item.Priority));
                System.Console.WriteLine(string.Format("RaiseTime: {0}", item.RaiseTime));
                System.Console.WriteLine(string.Format("ResetTime: {0}", item.ResetTime));
                System.Console.WriteLine(string.Format("SourceID: {0}", item.SourceID));
              System.Console.WriteLine(string.Format("SourceType: {0}", item.SourceType));
                System.Console.WriteLine(string.Format("State: {0}", item.State));
                System.Console.WriteLine(string.Format("StateText: {0}", item.StateText));
                System.Console.WriteLine(string.Format("SuppressionState: {0}",
item.SuppressionState));
                System.Console.WriteLine(string.Format("SystemSeverity: {0}",
item.SystemSeverity));
                System.Console.WriteLine(string.Format("TextColor: {0}", item.TextColor));
                System.Console.WriteLine(string.Format("User: {0}", item.UserName));
```

Copy code

```
                System.Console.WriteLine(string.Format("UserResponse: {0}",
item.UserResponse));
                System.Console.WriteLine(string.Format("Value: {0}", item.Value));
                System.Console.WriteLine(string.Format("ValueLimit: {0}",
item.ValueQuality));

                AlarmList.Add(item);
            }
            ...
        }
        ...
    }
}
```

See also

IAlarm (Page 7880)

IAlarmSubscription (Page 7895)

IAlarm

Description

The C# interface "IAlarm" specifies properties and methods for handling active alarms of the runtime system.

The methods trigger an exception in the case of an error.

Members

The following properties and methods are specified in the interface:

"Error" property

Error code of the alarm

```
uint32 Error { get; set; }
```

"Name" property

Name of the active alarm

```
string Name { get; set; }
```

"Acknowledge" method

Acknowledge active alarm or instance of an active alarm synchronously.

```
void Acknowledge(UInt32 instanceId)
```

- `instanceID`
  - Value "0": Acknowledge all instances of an active alarm.
  - Value > "0": Acknowledge instance with this ID.

"Disable" method

Deactivate generation of the alarm in the alarm source synchronously.

```
void Disable()
```

"Enable" method

Activate generation of the alarm in the alarm source again synchronously.

```
void Enable()
```

"Reset" method

Acknowledge the inactive state of an active alarm or an instance of an active alarm synchronously.

```
void Reset(UInt32 instanceId)
```

- `instanceID`
  - Value "0": Acknowledge the inactive state of the active alarm.
  - Value > "0": Acknowledge the inactive state of an instance with this ID.

"Shelve" method

Hide active alarm synchronously.

```
void Shelve()
```

"Unshelve" method

Display hidden alarm synchronously again.

```
void Unshelve()
```

Example

Acknowledge active alarm synchronously:

Copy code

```
public void AcknowledgeAlarms(IList<IAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        foreach (var alarm in AlarmList)
        {
            var alarmAck = runtime.GetObject<IAlarm>(alarm.Name);
            alarmAck.Acknowledge((alarm.InstanceID); // to acknowledge a particular alarm
instance, parameter can be ommitted if all instances of an alarm shall be acknowledged.
        }
    }
}
```

See also

IAlarmResult (Page 7872)

IAlarmSet (Page 7882)

IAlarmSubscription (Page 7895)

IAlarmSet

Description

The C# interface "IAlarmSet" specifies properties, methods and events for optimized access to several active alarms of the runtime system.

After the initialization of the "IAlarmSet" object, you can execute asynchronous operations with multiple alarms in one call, e. g. acknowledgment. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in the case of an error.

Members

The following properties, methods and events are only specified in the interface:

Access operator "this[string]"

Accessing an element of an alarm set via the alarm name.

```
IAlarm this[string alarmName] { get; set; }
```

`alarmName`
Name of the alarm that is changed in the AlarmSet.

"Count" property

Number of alarms of an AlarmSet list.

```
int32 Count { get; }
```

"Add" method

Add an active alarm or an instance of the alarm to the AlarmSet.

```
IAlarm Add(string alarmName, UInt32 instanceId)
```

- `alarmName`
  Name of the alarm that is added to the AlarmSet.
- `instanceId`
  Value = "0": Add all instances of an active alarm.
  Value > "0": Add instance with this ID.

"Remove" method

Remove a single alarm or an instance of an alarm from the AlarmSet.

```
void Remove(string alarmName, UInt32 instanceId)
```

- `alarmName`
  Name of the alarm that is removed from the AlarmSet.
- `instanceID`
  value = "0": Remove all instances of an active alarm
  Value > "0": Remove instance with this ID.

"Clear" method

Remove all alarms from an AlarmSet.

```
void Clear()
```

"Acknowledge" method

Acknowledge alarms of the AlarmSet asynchronously.

```
int Acknowledge()
```

"Disable" method

Deactivate generation of alarms of the AlarmSet in the alarm source asynchronously.

```
void Disable()
```

"Enable" method

Activate generation of alarms of the AlarmSet in the alarm source again asynchronously.

```
void Enable()
```

"Reset" method

Acknowledge inactive state of the alarms of the AlarmSet asynchronously. This method is relevant for alarms with the state machine "Alarm with acknowledgment and confirmation". Other state machines do not require a reset.

```
void Reset()
```

"Shelve" method

Hide alarms of the AlarmSet asynchronously.

```
void Shelve()
```

"Unshelve" method

Show hidden alarms of the AlarmSet asynchronously again.

```
void Unshelve()
```

"OnAcknowledgeHandler" event

When an AlarmSet is acknowledged with the "Acknowledge" and "AckknowledgeInstance" methods, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous acknowledgment of the alarms.

```
event OnAlarmSetEventHandler OnAcknowledgeHandler
```

"OnResetHandler" event

When the inactive state of the alarms of an AlarmSet is acknowledged with the "Reset" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous removal of the alarms.

```
event OnAlarmSetEventHandler OnResetHandler
```

"OnDisableHandler" event

When the alarms of an AlarmSet are deactivated with the "Disable" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous deactivation of the alarms.

```
event OnAlarmSetEventHandler OnDisableHandler
```

"OnEnableHandler" event

When the alarms of an AlarmSet are reactivated with the "Enable" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous reactivation of the alarms.

```
event OnAlarmSetEventHandler OnEnableHandler
```

"OnShelveHandler" event

When the alarms of an AlarmSet are hidden with the "Shelve" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the asynchronous hiding of the alarms.

```
event OnAlarmSetEventHandler OnShelveHandler
```

"OnUnshelveHandler" event

When the alarms of an AlarmSet are displayed again with the "Unshelve" method, the event calls an instance of the "OnAlarmSetEventHandler" delegate.

Declares the event and the event handler for the new asynchronous displaying of the alarms.

```
event OnAlarmSetEventHandler OnUnshelveHandler
```

"OnAlarmSetEventHandler" delegate

Specifies the signature of the event handling method for all events of an AlarmSet.

```
void OnAlarmSetEventHandler(
     IAlarmSet sender,
     uint32 errorCode,
     IList<IAlarmSetResult> values,
     bool completed)
```

- `sender`
  Source of the event
- `errorCode`
  Error during asynchronous transfer

- `values`
  Event data as a list of "IAlarmSetResult" objects of the processed alarms.
- `completed`
  Status of the asynchronous transfer:
  - True: All alarms of the AlarmSet are processed.
  - False: Not all alarms of the AlarmSet are processed yet.

Example

Acknowledge active alarms from the "AlarmList" list as an AlarmSet asynchronously:

Copy code

```
public void AcknowledgeAlarms(IList<IAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAlarm alarmAck = runtime.GetObject<IAlarm>(AlarmList[0].Name);
            alarmAck.Acknowledge();
        }
        else
        {
            AlarmAckList = new List<IAlarmResult>();
            IAlarmSet alarmSet = runtime.GetObject<IAlarmSet>(); ;
            foreach (var alarmResult in AlarmList)
            {
                IAlarm pAlarm = null;
                pAlarm = alarmSet.Add(alarmResult.Name);
                AlarmAckList.Add(alarmResult);
            }

            // Assign callback function
            alarmSet.OnAcknowledgeHandler += alarmSet_OnAcknowledgeHandler;
            alarmSet.Acknowledge();
        }
    }
}

public void alarmSet_OnAcknowledgeHandler(IAlarmSet sender, uint errorCode,
IList<IAlarmSetResult> values, bool completed)
{
    foreach (var item in values)
    {
       System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName: {2} ",
item.InstanceId, item.Name, item.SystemName));
    }
}
```

Remove active alarms from the "AlarmList" list as an AlarmSet asynchronously:

Copy code

```
public void ResetAlarms(IList<IAlarmResult> AlarmList)
{
    if (AlarmList != null)
    {
        if (AlarmList.Count == 1)
        {
            IAlarm alarmReset = runtime.GetObject<IAlarm>(AlarmList[0].Name);
            alarmReset.Reset();
        }
        else
        {
            IAlarmSet alarmSet = runtime.GetObject<IAlarmSet>(); ;
            AlarmResetList = new List<IAlarmResult>();
            foreach (var alarmResult in AlarmList)
            {
                IAlarm pAlarm = null;
                pAlarm = alarmSet.Add(alarmResult.Name);
                AlarmResetList.Add(alarmResult);
            }
            alarmSet.OnResetHandler += alarmSet_OnReseteHandler;
            try
            {
                alarmSet.Reset();
            }
            catch (OdkException ex)
            {
                System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
            }
        }
    }
}

public void alarmSet_OnReseteHandler(IAlarmSet sender, uint errorCode,
IList<IAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

IAlarmSetResult (Page 7887)

IAlarm (Page 7880)

IAlarmSetResult

Description

The C# interface "IAlarmSetResult" specifies properties of alarms in AlarmSets. These properties are returned by the EventHandler for all events of AlarmSets.

Members

The following properties are specified in the interface:

"SystemName" property

System name of the runtime system of an alarm in the AlarmSet

```
string SystemName { get; }
```

"Name" property

Name of an alarm in the AlarmSet

```
string Name { get; }
```

"InstanceId" property

InstanceID of an alarm in the AlarmSet

```
uint32 InstanceId { get; }
```

"Error" property

Error code of an alarm in the AlarmSet

```
uint32 Error { get; }
```

Example

Read out alarm of an AlarmSet after acknowledgment

Copy code

```
public void alarmSet_OnAcknowledgeHandler(IAlarmSet sender, uint errorCode,
IList<IAlarmSetResult> values, bool completed)
{
    try
    {
        foreach (var item in values)
        {
            System.Console.WriteLine(string.Format("InstanceId: {0} Name: {1} SystemName:
{2} ",
                item.InstanceId, item.Name, item.SystemName));
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

IAlarmSet (Page 7882)

IAlarmTrigger

Description

The C# interface "IAlarmTrigger" specifies methods for triggering alarms.

Members

"CreateSystemAlarm" method

Creates an alarm of the SystemAlarmWithoutClearEvent class with the state machine alarm without "Inactive with acknowledgment" status.

```
void CreateSystemAlarm(object alarmText, string area, object
alarmParameterValue1, ...,  object alarmParameterValue7)
```

- `alarmText`:
  The alarm text. You have the following options:
  - Transfer a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-defined text lists

    This method processes only user-defined text lists.

  - Transfer static string with monolingual text.
- `area`:
  The area of origin of the alarm
- `alarmParameterValue1` to `alarmParameterValue7`:
  User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEvent:SystemAlarmWithoutClearEvent`
- For monolingual alarm text:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEventText:SystemAlarmWithoutClearEvent`

"CreateSystemInformation" method

Creates an alarm of the SystemInformation class with the state machine alarm without "Inactive without acknowledgment" status.

```
void CreateSystemInformation(object alarmText, string area, object
alarmParameterValue1, ...,  object alarmParameterValue7)
```

- `alarmText`:
  The alarm text. You have the following options:
  - Transfer a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-defined text lists

    This method processes only user-defined text lists.

  - Transfer static string with monolingual text.
- `area`:
  The area of origin of the alarm
- `alarmParameterValue1` to `alarmParameterValue7`:
  User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts: `@ScriptingSystemEvents.SystemInformation:SystemInformation`
- For monolingual alarm text: `@ScriptingSystemEvents.SystemInformationText:SystemInformation`

"CreateOperatorInputInformation" method

Creates an alarm of the OperatorInputInformation class with the state machine alarm without "Inactive without acknowledgment" status.

```
void CreateOperatorInputInformation(object alarmText, string area,
object alarmParameterValue1, ...,  object alarmParameterValue7)
```

- `alarmText`:
  The alarm text. You have the following options:
  - Transfer a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-defined text lists

    This method processes only user-defined text lists.

  - Transfer static string with monolingual text.
- `area`:
  The area of origin of the alarm
- `alarmParameterValue1` to `alarmParameterValue7`:
  User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts:
  `@ScriptingSystemEvents.OperatorInputInformation:OperatorInputInformation`
- For monolingual alarm text:
  `@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation`

Example

The following code examples show how to trigger alarms of the class SystemAlarmWithoutClearEvent. Alarms of the classes SystemInformation and OperatorInputInformation are triggered in the same way.

Copying code

```
public void CreateSystemAlarm()
{
   //Create SystemAlarm with monolingual alarm text
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    systemAlarm.CreateSystemAlarm(alarmText: "Alarm Text", area: "Area",
        alarmParameterValue1: "Param1",
        alarmParameterValue2: "Param2",
        alarmParameterValue3: "Param3",
        alarmParameterValue4: "Param4",
        alarmParameterValue5: "Param5",
        alarmParameterValue6: "Param6",
        alarmParameterValue7: "Param7");
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}


public void CreateSystemAlarmWithAlarmTextAsTextList()
{
   //Create SystemAlarm with multilingual alarm text; the tranlsations are directly stored
in the text list
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
     var texlistforAlarmText = runtime.GetObject<ITextList>();
    // Text list: AlarmTextTemplate
    // Test list entry index: 101
    // Text: "My input msg. input value = @1%d@"
    texlistforAlarmText.Name = "AlarmTextTemplate";
    texlistforAlarmText.TextListEntryIndex = 101;
    systemAlarm.CreateSystemAlarm(alarmText: texlistforAlarmText, area: "Area",
        alarmParameterValue1: "125", // dynamic value for format specifier @1%d@
        alarmParameterValue2: "Param2",
        alarmParameterValue3: "Param3",
        alarmParameterValue4: "Param4",
```

Copying code

```
        alarmParameterValue5: "Param5",
        alarmParameterValue6: "Param6",
        alarmParameterValue7: "Param7");
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}


public void CreateSystemAlarmWithTextListAsParameterValue()
{
   //Create SystemAlarm with multilingual alarm text; the text list references other text
lists with tranlsations
    var alarm = runtime.GetObject<IAlarmSubscription>();
    var systemNames = new List<string>();
    systemNames.Add("SYSTEM1");
    alarm.OnAlarmHandler += alarm_OnAlarmHandler;
    alarm.OnPendingAlarmCompleteHandler += Alarm_OnPendingAlarmCompleteHandler;
    alarm.Filter = "AlarmClassName = 'SystemAlarmWithoutClearEvent'";
    alarm.Language = 1033;
    alarm.SystemNames = systemNames;
    alarm.Start();
    var systemAlarm = runtime.GetObject<IAlarmTrigger>();
    var textList1 = runtime.GetObject<ITextList>("Text_List_1");
    var textList2 = runtime.GetObject<ITextList>("Text_List_2");
    textList1.TextListEntryIndex = 1; //Eng TL @1%t#2T@ Val: @3%s@
    systemAlarm.CreateSystemAlarm(alarmText: textList1, area: "Area",
    alarmParameterValue1: 1,
    alarmParameterValue2: textList2, // text list object
    alarmParameterValue3: "Hello"); // Dynamic value of @3%s@
    _event.WaitOne();
    _event.Reset();
    //stop alarm subscription
    alarm.Stop();
    //Dispose alarm object
    alarm.Dispose();
}
```

See also

ITextList (Page 7895)

ITextList

Description

The C# interface "ITextList" is used to transfer multilingual alarm texts for system alarms and operator input alarms. See section IAlarmTrigger (Page 7888), CreateSystemInformation method. An ITextList instance is passed to the alarm text. When the operator input alarm is generated, it is replaced by the configured text from the text list.

Members

"Name" property

The name of the text list.

```
string Name { get; set; }
```

"TextListEntryIndex" property

The index of the list entry

```
UInt32 TextListEntryIndex { get; set; }
```

IAlarmSubscription

Description

The C# interface "IAlarmSubscription" specifies methods for monitoring alarms of the runtime system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following methods and events are specified in the interface:

"SystemNames" property

System names of runtime systems for the monitoring of active alarms

```
IList<string> SystemNames { get; set; }
```

"Language" property

Country identifier of the language of the monitored alarms

```
uint32 Language { get; set; }
```

"Filter" property

SQL-type string for filtering the result set of active alarms

```
string Filter { get; set; }
```

All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 7828).

"Start" method

Subscribe systems for monitoring changes of active alarms.

```
void Start()
```

"Stop" method

Clear monitoring of active alarms.

Note

Start and Stop in Windows Forms applications

Do not call the "Stop" method for a Windows forms application in the same thread in which you called "Start".

```
void Stop()
```

"OnAlarmHandler" event

Following a change of the alarm state on subscribed systems, the event calls an instance of the "OnAlarmHandler" delegate.

Declares the event and the event handler for asynchronous monitoring of alarms.

```
event OnAlarmHandler OnAlarmHandler
```

"OnPendingAlarmCompleteHandler" event

After completion of the handling of all active alarms of a system, the event calls an instance of the "OnPendingAlarmCompleteHandler" delegate.

Declares the event and the event handler for confirmation of the asynchronous operations.

```
event OnPendingAlarmCompleteHandler OnPendingAlarmCompleteHandler
```

"OnAlarmHandler" delegate

Specifies the signature of the event handling method for the "OnAlarmHandler" event.

```
void OnAlarmHandler(
     IAlarmSubscription sender,
     uint32 systemError,
     string systemName,
     IList<IAlarmResult> values,
     bool completed)
```

- `sender`
  Reference to the "IAlarmSubscription" object
- `systemError`
  Error code for the asynchronous operation
- `systemName`
  Name of the runtime system that is subscribed for alarm monitoring by the user.

- `values`
  Event data as a list of "IAlarmResult" objects of the monitored active alarms.
- `completed`
  Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are read out yet.

"OnPendingAlarmCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPendingAlarmCompleteHandler" event.

```
void OnPendingAlarmCompleteHandler(IAlarmSubscription sender)
```

- `sender`
  Source of the event

Example

Define functions for asynchronous monitoring of active alarms. Monitor "OnAlarmHandler" event and when occurring call the event handling method "alarm_OnAlarmHandler":

Copy code

```
public void SubscribeAlarmWithFilterOnOriginAndAlarmclass()
{
    try
    {
        //  object filter = "AlarmClass = 'AlarmWithOptionalAcknowledgement' AND Origin =
'Boiler'";
        IAlarmSubscription alarm = runtime.GetObject<IAlarmSubscription>();
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");

        // Assign alarm handler
        alarm.OnAlarmHandler += alarm_OnAlarmHandler;
        alarm.Filter = "AlarmClassName = 'AlarmWithOptionalAcknowledgement' AND Origin =
'Boiler'";
        alarm.Language = 1033;
        alarm.SystemNames = systemNames;
        alarm.Start();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

public void alarm_OnAlarmHandler(IAlarmSubscription sender, UInt32 nGlobalError, String
systemName, IList<IAlarmResult> value, bool completed)
{
    if (0 == nGlobalError)
    {
        try
        {
            AlarmList = new List<IAlarmResult>();
            foreach (var item in value)
            {
               System.Console.WriteLine(string.Format("InstanceID: {0} SourceID: {1} Name:
{2} State: {3} EventText: {4} StateText: {5} BackColor: {6} Flashing: {7} Quality: {8}
ModificationTime: {9}",
                    item.InstanceID, item.SourceID, item.Name, item.State,
                    item.EventText, item.StateText, item.BackColor, item.Flashing,
item.Quality, item.ModificationTime));
                AlarmList.Add(item);
            }

            // For test purpose: Cancel subscription after first notification
            AcknowledgeAlarms(AlarmList);
        }
        finally
        {
            if (null != sender)
            {
                sender.Stop();
                sender.Dispose();
            }
        }
    }
```

Copy code

```
    else
    {
        System.Console.WriteLine("AlarmSubscription Failed");
    }
}
```

See also

IAlarmResult (Page 7872)

IAlarm (Page 7880)

ILoggedAlarmResult

Description

The C# interface "ILoggedAlarmResult" specifies methods for accessing properties of logged alarms of a logging system.

Members

The following properties are specified in the interface:

"InstanceID" property

InstanceID for a logged alarm with multiple instances

```
uint32 InstanceID { get; }
```

"Name" property

Name of the logged alarm

```
string Name { get; }
```

"AlarmClassName" property

Name of the alarm class of the logged alarm

```
string AlarmClassName { get; }
```

"AlarmClassSymbol" property

Symbol of the alarm class of the logged alarm

```
string AlarmClassSymbol { get; }
```

"AlarmParameterValues" property

Parameter values of the logged alarm

```
IReadOnlyList<object> AlarmParameterValues { get; }
```

"AlarmText1" ... "AlarmText9" properties

Additional texts 1-9 of the logged alarm

```
string AlarmText1 { get; }
...
string AlarmText9 { get; }
```

"ChangeReason" property

Trigger event for modification of the alarm state of the logged alarm

```
uint16 ChangeReason { get; }
```

"Connection" property

Connection via which the logged alarm was triggered

```
string Connection { get; }
```

"State" property

Alarm state of the logged alarm

```
HmiAlarmState State { get; }
```

The enumeration "HmiAlarmState" can contain the following values:

- Standard (0x00)
- Raised (0x01)
- RaisedCleared (0x02)
- RaisedAcknowledged (0x05)
- RaisedAcknowledgedCleared (0x06)
- RaisedClearedAcknowledged (0x07)
- Removed (0x80)

"StateText" property

Alarm state of the logged alarm as text, for example, "active" or "inactive"

```
string StateText { get; }
```

"EventText" property

Text of the logged alarm that describes the alarm event

```
string EventText { get; }
```

"TextColor" property

Text color of the alarm state of the logged alarm

```
uint32 TextColor { get; }
```

"BackColor" property

Background color of the alarm state of the logged alarm

```
uint32 BackColor { get; }
```

"Flashing" property

Indicates whether a logged alarm is flashing.

```
bool Flashing { get; }
```

"InvalidFlags" property

Identification of the logged alarm with invalid data

```
byte InvalidFlags { get; }
```

"ModificationTime" property

Time of the last modification to the alarm state of the logged alarm

```
DateTime ModificationTime { get; }
```

"RaiseTime" property

Trigger time of the logged alarm

```
DateTime RaiseTime { get; }
```

"AcknowledgementTime" property

Time of alarm acknowledgment of the logged alarm

```
DateTime AcknowledgementTime { get; }
```

"ClearTime" property

Time of reset of the logged alarm

```
DateTime ClearTime { get; }
```

"ResetTime" property

Time of reset of the logged alarm

```
DateTime ResetTime { get; }
```

"SuppressionState" property

Status of visibility of the logged alarm

```
byte SuppressionState { get; }
```

"SystemSeverity" property

Severity of the system error

```
byte SystemSeverity { get; }
```

"Priority" property

Relevance for display and sorting of the logged alarm

```
byte Priority { get; }
```

"Origin" property

Origin for display and sorting of the logged alarm

```
string Origin { get; }
```

"Area" property

Origin area for display and sorting of the logged alarm

```
string Area { get; }
```

"Value" property

Process value of the logged alarm

```
object Value { get; }
```

"AlarmGroupName" property

Name of the group to which the alarm belongs. Blank if the alarm does not belong to a group.

```
string AlarmGroupName { get; }
```

"DeadBand" property

Range of the triggering tag in which no alarms are generated.

```
object DeadBand { get; }
```

"HostName" property

Name of the host that triggered the alarm.

```
string HostName { get; }
```

"InfoText" property

Localizable text for the alarm that contains the associated work instruction.

```
string InfoText { get; }
```

"StateMachine" property

StateMachine model of the alarm. The StateMachine represents the behavior of alarms through the arrangement of alarm states and alarm events, e.g. "RaiseClear", "RaiseRequiresAcknowledgment" or "RaiseClearOptionalAcknowledgment".

```
byte StateMachine { get; }
```

"ValueQuality" property

Quality of the process value of the alarm

```
int32 ValueQuality { get; }
```

"ValueLimit" property

Process value limit of the logged alarm

```
object ValueLimit { get; }
```

"SingleAcknowledgement" property

Indicates whether an alarm may be acknowledged only individually or may be acknowledged as a group or multiple selection.

```
bool SingleAcknowledgement { get; }
```

"LoggedAlarmStateObjectID" property

ID of alarm state for referencing within the logging system.

```
string LoggedAlarmStateObjectID { get; }
```

"ID" property

User-defined ID of the alarm that is also used in the display.

```
uint32 ID { get; }
```

"SourceType" property

Source from which the alarm was generated, for example, tag-based, controller-based or system-based alarm.

```
HmiAlarmSourceType SourceType { get; }
```

The enumeration "HmiAlarmSourceType" can contain the following values:

- `Undefined` (0)
- `Tag` (1)
- `Controller` (2)
- `System` (3)
- `Alarm` (4)

"UserName" property

User name of the logged operator input alarm

```
string UserName { get; }
```

"UserResponse" property

Expected or required user response to the logged alarm

```
byte UserResponse { get; }
```

"LoopInAlarm" property

Function that navigates from the alarm control to its origin.

```
string LoopInAlarm { get; }
```

"Error" property

Error code of the logged alarm

```
unit32 Error { get; }
```

"Duration" property

Returns the time interval in nanoseconds between triggering of the logged alarm and its previous status change.

```
TimeSpan  Duration { get; }
```

See also

IAlarmLogging (Page 7905)

IAlarmLoggingSubscription (Page 7907)

IAlarmLogging

Description

The C# interface "IAlarmLogging" specifies properties and methods for reading out logged alarms of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following methods and events are specified in the interface:

"Read" method

Read out logged alarms of a time period synchronously from logging system.

```
IList<ILoggedAlarmResult> Read(
     DateTime begin,
     DateTime end,
     string filter,
     uint32 language,
     IList<string> systemNames)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `filter`
  Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  Country identification of the language of the logged alarm text
- `systemNames`
  System names of the runtime systems of the logged alarms. Default: local system

"ReadAsync" method

Read out logged alarms of a time period asynchronously from logging system.

```
void ReadAsync(
     DateTime begin,
     DateTime end,
     string filter,
```

```
    uint32 language,
    IList<string> systemNames)
```

- `begin`
  Start date of the time period
- `end`
  End date of the time period
- `filter`
  Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  Country identification of the language of the logged alarm text
- `systemNames`
  System names of the runtime systems of the logged alarms. Default: local system

"OnReadComplete" event

After readout of the logged alarms, the event calls an instance of the "OnReadCompleteAlarmLoggingHandler" delegate.

Declares the event and the event handler for the asynchronous readout of logged alarms.

```
event OnReadCompleteAlarmLoggingHandler OnReadComplete
```

"OnReadCompleteAlarmLoggingHandler" delegate

Specifies the signature of the event handling method for the "OnReadComplete" event.

```
void OnReadCompleteAlarmLoggingHandler(
     IAlarmLogging sender,
     uint32 errorCode,
     IList<ILoggedAlarmResult> values,
     bool completed)
```

- `sender`
  Source of the event
- `errorCode`
  Error code for the asynchronous operation
- `values`
  Event data as a list of "ILoggedAlarmResult" objects of the read-out logged alarms.
- `completed`
  Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are read out yet.

Example

Reading out and outputting logged alarms asynchronously:

Copy code

```
public void ReadAsync()
{
    try
    {
        var alarm = _runtime.GetObject<IAlarmLogging>();
        var now = DateTime.UtcNow;
        var begin = now.AddMinutes(-5);
        var end = now.AddMinutes(-2);
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");
        alarm.OnReadComplete += Alarm_OnReadComplete;
        alarm.ReadAsync(begin, end, "", 1033, systemNames);
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

private void Alarm_OnReadComplete(IAlarmLogging sender, uint errorCode,
IList<ILoggedAlarmResult> values, bool completed)
{
    PrintValues(values);
    sender.Dispose();
}

private void PrintValues(IList<ILoggedAlarmResult> values)
{
    foreach (var v in values)
    {
        Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}", v.Name,
v.RaiseTime, v.Value));
    }
}
```

See also

ILoggedAlarmResult (Page 7900)

IAlarmLoggingSubscription (Page 7907)

IAlarmLoggingSubscription

Description

The C# interface "IAlarmLoggingSubscription" specifies properties and methods for monitoring logged alarms of a logging system.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"SystemNames" property

System names of the runtime systems of the logged alarms

```
IList<string> SystemNames { get; set; }
```

"Language" property

Country identification of the language of the logged alarms

```
uint32 Language { get; set; }
```

"Filter" property

SQL-type string for filtering the result set of the logged alarms

```
string Filter { get; set; }
```

"Start" method

Start monitoring of logged alarms.

```
void Start()
```

"Stop" method

Stop monitoring of all logged alarms.

```
void Stop()
```

"OnDataChanged" event

Following a change of a monitored alarm in the logging systems, the event calls an instance of the "OnDataChangedAlarmLoggingHandler" delegate.

Declares the event and the event handler for the monitoring of logged alarms.

```
event OnDataChangedAlarmLoggingHandler OnDataChanged
```

"OnDataChangedAlarmLoggingHandler" delegate

Specifies the signature of the event handling method for the "OnDataChanged" event.

```
void OnDataChangedAlarmLoggingHandler(
     IAlarmLoggingSubscription sender,
```

```
    uint32 errorCode,
    IList<ILoggedAlarmResult> values)
```

- `sender`
  Source of the event
- `errorCode`
  Error code for the asynchronous operation
- `values`
  Event data as a list of "ILoggedAlarmResult" objects of the monitored logged alarms.

Example

Output logged alarms following a change:

Copy code

```
public void SubscribeAlarm()
{
    try
    {
        _alarm = _runtime.GetObject<IAlarmLoggingSubscription>();
        List<string> systemNames = new List<string>();
        systemNames.Add("SYSTEM1");
        // Assign alarm handler
        _alarm.OnDataChanged += Alarm_OnDataChanged;
        _alarm.Filter = "";
        _alarm.Language = 1033;
        _alarm.SystemNames = systemNames;
        _alarm.Start();
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}

private void Alarm_OnDataChanged(IAlarmLoggingSubscription sender, uint errorCode,
IList<ILoggedAlarmResult> values)
{
    PrintValues(values);
}

private void PrintValues(IList<ILoggedAlarmResult> values)
{
    foreach (var v in values)
    {
        Console.WriteLine(string.Format("Name: {0} Timestamp: {1} Value: {2}", v.Name,
v.RaiseTime, v.Value));
    }
}
```

Cancel monitoring of logged alarms:

Copy code

```
public void CancelSubscription()
{
    try
    {
        if (_alarm != null)
        {
            _alarm.Stop();
            _alarm.Dispose();
        }
    }
    catch(Exception ex)
    {
        System.Console.WriteLine(ex.Message);
    }
}
```

See also

IAlarmLogging (Page 7905)

ILoggedAlarmResult (Page 7900)

20.3.8.6 Interfaces for connections

IConnectionResult

Description

The C# interface "IConnectionResult" provides access to the details of a connection.

Members

The following properties are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; }
```

"ConnectionState" property

Status of the connection

```
HmiConnectionState ConnectionState { get; }
```

The "HmiConnectionState" enumeration contains the following values:

- `Disabled` (0)
- `Connecting` (1)

- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"EstablishmentMode" property

Mode in which the connection will be established.

```
HmiConnectionEstablishmentMode EstablishmentMode { get; }
```

The "HmiConnectionEstablishmentMode" enumeration contains the following values:

- `None` (0)
- `AutomaticActive` (1)
- `AutomaticPassive` (2)
- `OnDemandActive` (3)
- `OnDemandPassive` (4)

"TimeSynchronizationMode" property

Mode of time synchronization between HMI system and automation system

```
HmiTimeSynchronizationMode TimeSynchronizationMode { get; }
```

The "HmiTimeSynchronizationMode" enumeration contains the following values:

- `None` (0)
- `Subordinate` (1)
- `Lead` (2)

"DisabledAtStartup" property

Indicates whether the connection is disabled at the start of Runtime.

```
bool DisabledAtStartup { get; }
```

- `true`: Connection is disabled at the connection start.
- `false`: Connection is active at the connection start.

"Enabled" property

Indicates whether the connection is active.

```
bool Enabled { get; }
```

- `true`: Connection is active.
- `false`: Connection is not active.

"ConnectionType" property

Protocol of a communication driver, e.g. "S7 Classic".

```
string ConnectionType { get; }
```

"Error" property

Error code of the connection

```
uint32 Error { get; }
```

Example

Output connection details:

Copy code

```
public void Connection_Read()
{
    var (connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    if (connection != null)
    {
        var con = connection.Read();
        if (con != null)
        {
            Console.WriteLine("Connection Name is  {0} ", con.Name);
            Console.WriteLine("ConnectionState is  {0}", con.ConnectionState);
           Console.WriteLine("TimeSynchronizationMode is {0} ", con.TimeSynchronizationMod
e);
            Console.WriteLine("EstablishMentMode is  {0} ", con.EstablishmentMode);
            Console.WriteLine("Enabled is  {0} ", con.Enabled);
            Console.WriteLine("DisabledAtStartup is  {0} ", con.DisabledAtStartup);
            Console.WriteLine("ConnectionType is  {0} ", con.ConnectionType);
            Console.WriteLine(" Error is {0} ", con.Error);
        }
    }
}
```

See also

IConnection (Page 7913)

IConnectionSet (Page 7915)

IConnectionStatusResult

Description

The C# interface "IConnectionStatusResult" provides access to the status of a connection.

Members

The following properties are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; }
```

"ConnectionState" property

Status of the connection

```
HmiConnectionState ConnectionState { get; }
```

The "HmiConnectionState" enumeration contains the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"Error" property

Error code of the connection

```
uint32 Error { get; }
```

Example

Output status of a certain connection:

Copy code

```
public void Connection_GetConnectionState()
{
    var connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    var con = connection.GetConnectionState();
    if (con != null)
    {
        Console.WriteLine("Connection Name is : {0} ", con.Name);
        Console.WriteLine(" Error is : {0} ", con.Error);
        Console.WriteLine("Connection status is: {0}", con.ConnectionStatus);
    }
}
```

See also

IConnection (Page 7913)

IConnectionSet (Page 7915)

IConnection

Description

The C# interface "IConnection" provides properties and methods for the access to a connection. A connection is a configured, logical assignment of two communication partners.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The following properties and methods are specified in the interface:

"Name" property

Name of the connection

```
string Name { get; set; }
```

"Read" method

Read connection details synchronously from the Runtime system.

```
IConnectionResult Read()
```

"GetConnectionState" method

Return connection status of a connection.

```
IConnectionStatusResult GetConnectionState()
```

"SetConnectionMode" method

Change connection status of a connection.

```
void SetConnectionMode(ConnectionMode mode)
```

The "ConnectionMode" enumeration contains the following values:

- `Disabled` (0)
- `Enabled` (1)

Examples

Disable connection:

Copy code

```
public void Connection_SetConnectionStateDisable()
{
    var connection = m_runtime.GetObject<IConnection>("HMI-ConnectionS7Plus");
    if (connection != null)
    {
        connection.SetConnectionMode(ConnectionMode.Disabled);
    }
}
```

Enable connection:

Copy code

```
public void SetConnectionStateEnable()
{
    using (IConnection connection = runtime.GetObject<IConnection>("HMI-ConnectionS7Plus"))
    {
        if (connection != null)
        {
            connection.SetConnectionMode(ConnectionMode.Enabled);
        }
    }
}
```

See also

IConnectionSet (Page 7915)

IConnectionResult (Page 7910)

IConnectionStatusResult (Page 7912)

IConnectionSet

Description

The C# interface "IConnectionSet" specifies properties, methods and events for optimized access to several connections of the Runtime system.

After initialization of the "IConnectionSet" object, you have read/write access to multiple connections in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple connections.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The following properties, methods and events are specified in the interface:

"ContextId" property

ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

Default value -1: The ContextId is not used.

```
int32 ContextId { get; set; }
```

Access operator "this[string]"

Accessing or changing an element of a ConnectionSet via the connection name.

```
IConnection this[string connectionName] { get; set; }
```

`connectionName`
Name of the connection

"Count" property

Number of connections of a connection set list

```
int32 Cont { get; }
```

"Add" method

Add connections to a connection set.

```
void Add(ICollection<string> connections)
```

`connections`
List of connections for the connection set

"Remove" method

Remove individual connection from connection set.

```
void Remove(string connection)
```

`connection`
Name of connection that is removed from the connection set.

"Clear" method

Remove all connections from connection set.

```
void Clear()
```

"Read" method

Read connection details of all connections of the connection set synchronously from the Runtime system.

```
IList<IConnectionResult> Read()
```

"ReadAsync" method

Read connection details of all connections of the connection set asynchronously from the Runtime system.

```
void ReadAsync()
```

"GetConnectionState" method

Read connection status of all connections of the connection set synchronously from the Runtime system.

```
IList<IConnectionStatusResult> GetConnectionState()
```

"Subscribe" method

Subscribe all connections of a connection set asynchronously for change monitoring.

```
void Subscribe()
```

"CancelSubscribe" method

Cancel change monitoring of all connections of a connection set.

```
void CancelSubscribe()
```

"OnConnectionRead" event

After establishment of the connection, event calls an instance of the "OnConnectionHandler" delegate.

Declares the event and the event handler for the establishment of a connection.

```
event OnConnectionHandler OnConnectionRead
```

"OnConnectionStateChange" event

After change of the connection status, event calls an instance of the "OnConnectionStateChangeHandler" delegate.

Declares the event and the event handler for the change of the connection status.

```
event OnConnectionStateChangeHandler OnConnectionStateChange
```

"OnConnectionHandler" delegate

Specifies the signature of the event handling method for the "OnConnectionHandler" event of a connection set.

```
void OnConnectionHandler(
     IConnectionSet sender,
     uint32 systemError,
     IList<IConnectionResult> values)
```

- `sender`
  Source of the event
- `systemError`
  Error code
- `values`
  List of connections

"OnConnectionStateChangeHandler" delegate

Specifies the signature of the event handling method for the "OnConnectionStateChangeHandler" event of a connection set.

```
void OnConnectionStateChangeHandler(
    IConnectionSet  sender,
    uint32 systemError,
    IList<IConnectionStatusResult> values)
```

- `sender`
  Source of the event
- `systemError`
  Error code
- `values`
  List of status changes of the connections

Examples

Monitor connection status:

Copy code

```
public void ConnnectionSet_Subscribe()
{
    Console.WriteLine(" ......Connection Set:Subscription start ......");
    using (IConnectionSet subscribe = runtime.GetObject<IConnectionSet>())
    {
        if (subscribe != null)
        {
            ICollection<string> list = new string[] { "HMI-Connection", "HMI-
ConnectionS7Plus" };
            subscribe.Add(list);
            subscribe.OnConnectionStateChanged += Subscribe_OnConnectionStateChanged;
            subscribe.Subscribe();
            Thread.Sleep(500);
        }
    }
}

private void Subscribe_OnConnectionStateChanged(IConnectionSet sender, uint systemError,
IList<IConnectionStatusResult> values)
{
    if (0 == systemError)
    {
        try
        {
            foreach (var item in values)
            {
                Console.WriteLine("Name:{0} ", item.Name);
                Console.WriteLine("State:{0} ", item.ConnectionStatus.ToString());
                Console.WriteLine("Error:{0} ", item.Error);
            }
        }

        catch (OdkException ex)
        {
        System.Console.WriteLine(string.Format("OdkException occured {0}", ex.Message));
        }

        finally
        {
            if (null != sender)
            {
                sender.CancelSubscribe();
                Console.WriteLine("Subscription cancelled");
                sender.Dispose();
            }
        }
    }
    else
    {
        System.Console.WriteLine("Connection set subscription  Failed");
    }
}
```

Copy code

Read out connection synchronously:

Copy code

```
public void ConnectionSet_Read()
{
    Console.WriteLine(" .......Connection Set: Read ...... ");
    using (IConnectionSet read = runtime.GetObject<IConnectionSet>())
    {
        if (read != null)
        {
            ICollection<string> list = new string[] { "HMI-Connection", "HMI-
ConnectionS7Plus" };
            read.Add(list);

            IList<IConnectionResult> connectionResult = read.Read();
            foreach (var item in connectionResult)
            {
                System.Console.WriteLine(string.Format("Connection Name is  {0} ",
item.Name));
                System.Console.WriteLine(string.Format("ConnectionState is  {0}",
item.ConnectionState.ToString()));
                System.Console.WriteLine(string.Format("TimeSynchronizationMode is {0} ",
item.TimeSynchronizationMode.ToString()));
                System.Console.WriteLine(string.Format(" Error is {0} ", item.Error));
                System.Console.WriteLine(string.Format("EstablishMentMode is  {0} ",
item.EstablishmentMode));
                System.Console.WriteLine(string.Format("Enabled is  {0} ", item.Enabled));
                System.Console.WriteLine(string.Format("DisabledAtStartup is  {0} ",
item.DisabledAtStartup));
                System.Console.WriteLine(string.Format("ConnectionType is  {0} ",
item.ConnectionType));
            }
        }
    }
}
```

Read out connection asynchronously:

Copy code

```
public void ConnectionSet_ReadAsync()
{
    Console.WriteLine(" ......Connection Set: ReadAsync start ......");
    IConnectionSet readAsync = runtime.GetObject<IConnectionSet>();
    if (readAsync != null)
    {
       ICollection<string> list = new string[] { "HMI-Connection", "HMI-ConnectionS7Plus" };
        readAsync.Add(list);
        readAsync.OnConnectionRead += Read_OnConnectionComplete;
        readAsync.ReadAsync();
        Thread.Sleep(5000);
    }
}

private void Read_OnConnectionComplete(IConnectionSet sender, uint systemError,
IList<IConnectionResult> values)
{
    foreach (var item in values)
    {
        Console.WriteLine("Name:{0} ", item.Name);
        Console.WriteLine("State:{0} ", item.ConnectionState);
        Console.WriteLine("establishmentMode:{0} ", item.EstablishmentMode);
        Console.WriteLine("TimeSynchronizationMode:{0} ", item.TimeSynchronizationMode);
        Console.WriteLine("ConnectionType:{0} ", item.ConnectionType);
        Console.WriteLine("Enabled:{0} ", item.Enabled);
        Console.WriteLine("DisabledAtStartup:{0} ", item.DisabledAtStartup);
        Console.WriteLine("Error:{0} ", item.Error);
    }
}
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

See also

IConnection (Page 7913)

IConnectionResult (Page 7910)

IConnectionStatusResult (Page 7912)

20.3.8.7 Interfaces of the Plant Model

IPlantModel

Description

The C# interface "IPlantModel" specifies methods for access to object instances of the plant model of a Runtime system. The "IPlantModel" object represents the plant model of the graphical Runtime system.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Formatting a path in the hierarchy

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Note

Fixed code HierarchyName

In the current version, `HierarchyName` has a fixed code which is "hierarchy".

Members

The class implements the following methods:

"GetPlantObject" method

Supplies an object instance of "IPlantObject".

```
IPlantObject GetPlantObject(string  plantObject)
```

- `plantObject`
  Identifies an `IPlantObject` instance by its name or its path in the hierarchy.

"GetPlantObjectsByType" method

Supplies a list with instances of "IPlantObject" that have a specific type.

```
IList<IPlantObject> GetPlantObjectsByType(string
plantObjectTypeFilter, string viewFilter = null)
```

- `plantObjectTypeFilter`
  Filter for the "IPlantObject" type on which the instances are based.
- Optional: `viewFilter`
  Filter for the path in the hierarchy. Only instances from a specific node are returned.

"GetPlantObjectsByExpression" method

Supplies a list with instances of "IPlantObject" instances. The instances are filtered by type and property values.

```
IList<IPlantObject> GetPlantObjectsByExpression(ICollection<string>
propertyNames, string plantObjectTypeFilter, string
expressionFilter, string viewFilter)
```

- `propertyNames`
  A list with property names
- `plantObjectTypeFilter`
  Filter for the object type on which the instances are based.
- `expressionFilter`
  An expression that is a filter for the property values.
- Optional: `viewFilter`
  Filter for a hierarchy path.

Example:

```
var plantObjectArr =
PlantModel.GetPlantObjectsByExpression("Temperature", "Motor",
"Temperature>100");
```

"GetPlantObjectsByPropertyNames" method

Supplies a list with "IPlantObject" instances that have specific properties and originate in a specific plant node.

```
IList<IPlantObject>
GetPlantObjectsByPropertyNames(ICollection<string> propertyNames,
string viewFilter = null)
```

- `propertyNames`
  A list with property names
  If the list contains multiple values, all properties must be available at the object.
- Optional: `viewFilter`
  Filter for a hierarchy path.

Example

Example of a hierarchy path:

Hierarchy path	Referenced object instance
`System2.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of system2.
`.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of the local system.
`U4711`	References the "U4711" object instance of the local system.
`System2::U4711`	References the "U4711" object instance of System2.

Sample code

Copy code

```
public void Odk_GetPlantObjectsByType()
{
    using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
    {    //gets node for specified Node path
        IList<IPlantObject> plantObject =
myPlantModel.GetPlantObjectsByType("RUNTIME_1::NodeType1",
        ".hierarchy::RootNodeName\\Node1");

        if (plantObject.Count() > 0)
        {
             foreach (IPlantObject item in plantObject)
             {
                 System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
item.CurrentPlantView, item.Name));
             }
        }
    }
}
```

IPlantObject

Description

The C# interface "IPlantObject" specifies properties and methods for handling object instances of the plant model of a Runtime system.

An object instance in the plant model is based on an object type and its data structure. Each object instance receives its position within the hierarchy by assigning it to a hierarchy node.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

`[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID`

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following properties, methods and events are specified in the interface:

"Name" property

The name for unique identification of the object instance.

```
string Name { get; }
```

"Parent" property

The parent object instance in the hierarchy.

```
IPlantObject Parent { get; }
```

"Children" property

List of child object instances in the hierarchy

```
IReadOnlyList<IPlantObject> Children { get; }
```

"PlantViewPaths" property

The dictionary with string/string pairs that maps the hierarchy names to the hierarchy paths for all hierarchies that contain the "IPlantObject" instance (hierarchy name to hierarchy path).

```
IReadOnlyDictionary<string, string> PlantViewPaths { get; }
```

"CurrentPlantView" property

Path and name of the object instance in the currently selected hierarchy, e.g. "Maintenance".

The "CurrentPlantView" property is used as basis for navigation with the "Parent" or "Children" properties. If the object instance is only contained in one hierarchy, "CurrentPlantView" contains its path. If the object is contained in several views, the hierarchy path must be set via this property before the "Parent" or "Children" property can be used.

```
string CurrentPlantView { get; set; }
```

"GetProperty" method

Supplies a property of the object instance.

```
IPlantObjectProperty GetProperty(string propertyName)
```

`propertyName`
Name of an object instance property

"GetProperties" method

Supplies a two-dimensional list (name-object pairs) of the data structure of the object instance. The list allows access to the instance properties.

```
IPlantObjectPropertySet GetProperties(ICollection<string>
propertyNames = null)
```

- Optional: `propertyNames`
  List with names of one or multiple object instance properties.
  Without parameters, all properties of an object instance are returned

"GetActiveAlarms" method

Supplies all active alarms that the object instance contains at the time it is called in the active hierarchy. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call. Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

```
void GetActiveAlarms(UInt32 languageId, bool includeChildren =
false, string filter = null);
```

- `languageID`
  Language code of the language for all texts of an alarm and the filters. Refer to the section Locale IDs of the supported languages (Page 7829).
- Optional: `includeChildren`
  The active alarms of the child instances are returned as well.
- Optional: `filter`
  SQL-type string for filtering the alarm texts. The filter can contain operators. See also Syntax of the alarm filter (Page 7828).

"CreateAlarmSubscription" method

Supplies a "PlantObjectAlarmSubscription" that can be used to start and stop an alarm subscription.

```
IPlantObjectAlarmSubscription CreateAlarmSubscription();
```

"PlantObjectAlarmHandler" event

The event calls an instance of the "OnPlantObjectAlarmHandler" delegate.

```
event OnPlantObjectAlarmHandler PlantObjectAlarmHandler;
```

"OnPlantObjectAlarmHandler" delegate

Specifies the signature of the event handling method for the "`PlantObjectAlarmHandler`" event of an "IPlantObject" instance.

```
public delegate void OnPlantObjectAlarmHandler(
        IPlantObject sender,
        UInt32 systemError,
        string systemName,
        IList<IAlarmResult> values,
        bool completed);
```

- `sender`
  Source of the event
- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `systemName`
  Name of the runtime system that is subscribed for alarm monitoring by the user.
- `values`
  Event data as a list of "IAlarmResult" instances of the monitored active alarm.
- `completed`
  Status of the asynchronous transfer:
  - `True`: All alarms are read out.
  - `False`: Not all alarms are read out yet.

Example

Copy code

```
public void Odk_PlantObjectGetProperties()
{
   var myPlantModel = _runtime.GetObject<IPlantModel>();
   var strNodeName = ".hierarchy::RootNodeName/Node1";
   var plantObject = myPlantModel.GetPlantObject(strNodeName);
   Console.WriteLine("ViewName: {0} Name: {1}", plantObject.CurrentPlantView, plantObject.Name);
   // get the  plant objectproperties by propeyty names
   var plantObjectProperties = plantObject.GetProperties();
   if (plantObjectProperties != null)
   {
      var nCount = plantObjectProperties.Count;
      var listPropValues = plantObjectProperties.Read();
      Console.WriteLine("Number of Properties {0}", nCount);
      foreach (var item in listPropValues)
      {
         Console.WriteLine("Property Name is {0}  ", item.Name);
         Console.WriteLine("Property Value is {0}  ", item.Value);
         Console.WriteLine("Property Quality is {0} ", item.Quality);
         Console.WriteLine("Property Error is {0} ", item.Error);
      }
   }
}
```

IPlantObjectProperty

Description

The C# interface "IPlantObjectProperty" specifies the handling of properties of object instances of the plant model of a Runtime system. The properties represent the data structure of an object instance.

The object instance communicates with the automation system through the properties of the data structure. The values of the properties are obtained from linked process tags or internal tags.

You reference an "IPlantObjectProperty" object using the `IPlantObject.GetProperty` or `IPlantObject.GetProperties` method.

The interface inherits the `Dispose` method of the "IDisposable" interface of the .NET framework.

Members

The interface has the following properties and methods:

"Name" property

Name of the property

```
string Name { get; }
```

"Read" method

Reads the value of the "IPlantObjectProperty" instance synchronously and returns it as an "IPlantObjectPropertyValue" object. The value, the quality code and the time stamp of the property are determined when the property is read.

```
IPlantObjectPropertyValue Read()
```

"Write" method

Writes the value synchronously to the "IPlantObjectProperty" instance.

```
void Write(object Value)
```

- `Value`
  New process value of the property

Example

Copy code

```
public void Odk_PlantObjectGetPropertyWrite()
{
    using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
    {
       string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        using (IPlantObject plantobject = myPlantModel.GetPlantObject(strNodeName))
        {
            System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantobject.CurrentPlantView, plantobject.Name));
            if (plantobject != null)
                {
                    string strName = "NodeProperty_1";
                    using (IPlantObjectProperty plantObjectProperty =
plantobject.GetProperty(strName))
                    {
                        if (plantObjectProperty != null)
                        {
                            IPlantObjectPropertyValue pValue = plantObjectProperty.Read();
                            System.Console.WriteLine(string.Format("Property Name: {0}
property value before write
                            operation: {1}", strName, pValue.Value));
                            Object value = 400; // Write Cpm Node Property
                            plantObjectProperty.Write(value);
                           IPlantObjectPropertyValue pValues = plantObjectProperty.Read();
                            System.Console.WriteLine(string.Format("Property Name: {0}
property value after write Operration: {1}",strName, pValues.Value));
                        }
                    }
                }
            }
        }
    }
}
```

IPlantObjectPropertyValue

Description

The C# interface "IPlantObjectPropertyValue" specifies the properties of process tags that are connected to an object instance property of the Runtime system.

Members

The following properties are specified in the interface:

"Name" property

Name of the tag

```
string Name { get; }
```

"Value" property

Value of the tag at the moment of the read operation.

```
object Value { get; }
```

"Quality" property

Quality code of the read operation of the tag.

```
Int32 Quality { get; }
```

"TimeStamp" property

Time stamp of the last successful read operation of the tag.

```
DateTime TimeStamp { get; }
```

"Error" method

Error code of the last read or write operation of the tag.

```
UInt32 Error { get; }
```

"OnPlantModelPropertySubscriptionHandler" delegate

Specifies the signature of the event handling method for the "`OnPlantModelPropertySubscriptionHandler`" event of an "IPlantObjectPropertySet" instance.

```
public delegate void OnPlantModelPropertySubscriptionHandler(
        IPlantObjectPropertySet sender,
         IList<IPlantObjectPropertyValue> values);
```

- `sender`
  Source of the event
- `values`
  Event data as a list of "IPlantObjectPropertyValue" instances of the monitored active alarm.

Example

Copy code

```
public void Odk_PlantObjectGetPropertyRead()
{
    using (var myPlantModel = _runtime.GetObject<IPlantModel>())
    {
        var strNodeName = ".hierarchy::RootNodeName/Node1";
        //gets node for specified Node path
        using (var plantobject = myPlantModel.GetPlantObject(strNodeName))
        {
            Console.WriteLine("ViewName: {0} Name: {1}", plantobject.CurrentPlantView, plantobject.Name);
            var strName = "NodeProperty_1";
            using (var plantObjectProperty = plantobject.GetProperty(strName))
            {
                if (plantObjectProperty != null)
                {
                    // Read Cpm Node Property
                    var plantObjectPropertyValue = plantObjectProperty.Read();
                    if (null != plantObjectPropertyValue)
                    {
                        Console.WriteLine(
                        "Name= {0} TimeStamp {1} QualityCode {2} Error {3} Value {4}",
                        plantObjectPropertyValue.Name, plantObjectPropertyValue.TimeStamp,
                        plantObjectPropertyValue.Quality, plantObjectPropertyValue.Error,
                        plantObjectPropertyValue.Value);
                    }
                }
            }
        }
    }
}
```

IPlantObjectPropertySet

Description

The C# interface "IPlantObjectPropertySet" specifies properties, methods and events for optimized access to several IPlantObjectProperty instances of the Runtime system.

After initialization of the "IPlantObjectPropertySet" object, you have read/write access to multiple IPlantObjectProperty instances in one call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Member

The following properties, methods and events are specified in the interface:

"ContextID" property

"ContextID" can be useful for asynchronous methods. Users can assign "ContextID" if they have to assign the answer from the system to a read/write job.

Default value -1: "ContextId" is not used.

```
Int32 ContextId { get; set; }
```

"[propertyName]" property

Change the process value of a property of the "IPlantObjectPropertySet" instance.
The value is changed by the property only in the local "IPlantObjectPropertySet" instance. To write the values in the process image, a "Write" or "WriteAsync" method must be called.

```
object this[string propertyName] { get; set; }
```

- `propertyName`
  Name of the property that is changed in the PropertySet.

"Count" property

The number of properties of the "IPlantObjectPropertySet" instance.

```
Uint32 Count { get; }
```

"Read" method

Supplies a list with all values of the "IPlantObjectProperty" instances contained in the "IPlantObjectPropertySet" instance. The values are read synchronously.

```
IList<IPlantObjectPropertyValue> Read();
```

"ReadAsync" method

Reads the values of all "IPlantObjectProperty" instances of the "IPlantObjectPropertySet" instance asynchronously.

```
void ReadAsync()
```

"Write" method

Writes the values of the "IPlantProperty" instances of the "PlantObjectPropertySet" instance synchronously to the Runtime system. Write operation errors are returned in a list with "IErrorResult" instances.

```
IList<IErrorResult> Write()
```

"WriteAsync" method

Writes the values of all "IPlantObjectProperty" instances of the "PlantObjectPropertySet" instance asynchronously to the Runtime system.

```
void WriteAsync()
```

"Subscribe" method

Subscribes all properties of the "IPlantObjectPropertySet" instance asynchronously for change monitoring.

```
void Subscribe()
```

"Add" method

Adds one or more "IPlantObjectProperty" instances to the "IPlantObjectPropertySet" instance.

The method can be called as follows:

- Adding multiple "IPlantObjectProperty" instances without value:
  `void Add(ICollection<string> propertyNames)`
  - `name`
    A collection with the names of the "IPlantObjectProperty" instances.
- Adding a "IPlantObjectProperty" instance with value:
  `void Add(string propertyName, object value);`
  - `propertyName`
    Name of the "IPlantObjectProperty" instance
  - `value`
    New process value of the "IPlantObjectProperty" instance

"Remove" method

Removes a property from the "IPlantObjectPropertySet" instance.

```
void Remove(string propertyName)
```

- `propertyName`
  Name of the property that is being removed.

"Clear" method

Removes all properties from the "IPlantObjectPropertySet" instance.

```
void Clear()
```

"OnPropertySetReadComplete" event

After completion of the read operation of the "ReadAsync" method, the event calls an instance of the "OnPropertySetReadCompleteHandler" delegate.

Declares the event and the event handler for asynchronous read operations.

```
event OnPropertySetReadCompleteHandler OnPropertySetReadComplete
```

"OnPropertySetWriteComplete" event

After completion of the write operation of the "WriteAsync" method, the event calls an instance of the "OnPropertySetWriteCompleteHandler" delegate.

Declares the event and the event handler for asynchronous write operations.

```
event OnPropertySetWriteCompleteHandler OnPropertySetWriteComplete
```

"OnPlantModelPropertySubscriptionNotification" event

After the change of a monitored PropertySet, the event calls an instance of the "OnPlantModelPropertySubscriptionHandler" delegate.

Declares the event and the event handler when changing a PropertySet.

```
event OnPlantModelPropertySubscriptionHandler
OnPlantModelPropertySubscriptionNotification
```

"OnPropertySetReadCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPropertySetReadComplete" event of an "IPlantObjectPropertySet" instance.

```
void OnPropertySetReadCompleteHandler(
        IPlantObjectPropertySet sender,
        UInt32 errorCode,
        IList<IPlantObjectPropertyValue> values)
```

- `sender`
  Source of the event
- `errorCode`
  Supplies an error code when a global error has occurred.
- `values`
  Event data as a list of "IPlantObjectPropertyValue" instances of the read property.

"OnPropertySetWriteCompleteHandler" delegate

Specifies the signature of the event handling method for the "OnPropertySetWriteComplete" event of an "IPlantObjectPropertySet" instance.

```
void OnPropertySetWriteCompleteHandler(
        IPlantObjectPropertySet sender,
        UInt32 errorCode,
        IList<IPlantObjectPropertyValue> values)
```

- `sender`
  Source of the event
- `errorCode`
  Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `values`
  Event data as a list of "IPlantObjectPropertyValue" instances of the read property.

"OnPlantModelPropertySubscriptionHandler" delegate

Specifies the signature of the event handling method for the "OnPlantModelPropertySubscriptionNotification" event of an "IPlantObjectPropertySet" instance.

```
void OnPlantModelPropertySubcriptionHandler(
        IPlantObjectPropertySet sender,
        IList<IPlantObjectPropertyValue> values)
```

- `sender`
  Source of the event
- `values`
  Event data as a list of the changed "IPlantObjectPropertyValue" instances.

Example

Copy code

```
public void Odk_PlantObjectGetPropertySetReadAsync()
{
    try
    {
        IPlantModel myPlantModel = runtime.GetObject<IPlantModel>();
       string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        IPlantObject plantObject = myPlantModel.GetPlantObject(strNodeName);
        System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantObject.CurrentPlantView,
        plantObject.Name));
        if (plantObject != null)
        {
            ICollection<string> PropNames = null;// get the plant objectproperties by
propeyty names
            IPlantObjectPropertySet plantObjectPropertyset =
plantObject.GetProperties(PropNames);
            if (plantObjectPropertyset != null)
            {
                plantObjectPropertyset.OnPropertySetReadComplete +=
odkPlantModel_onReadComplete; // Read Plant
                Object properties values asynchronously
                plantObjectPropertyset.ReadAsync();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
public void odkPlantModel_onReadComplete(IPlantObjectPropertySet sender, UInt32
SystemError, IList<IPlantObjectPropertyValue> Values)
{
    try
    {
        foreach (var value in Values)
        {
            Console.WriteLine("Name {0}", value.Name);
            Console.WriteLine("TimeStamp {0}", value.TimeStamp);Console.WriteLine("Value
{0}", value.Value);
           Console.WriteLine("Quality {0}", value.Quality);Console.WriteLine("Error {0}",
value.Error);
        }
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
}
```

Copy code

Copy code

```
public void odk_plantModelSubscribe()
{
    try
    {
        IPlantModel myPlantmodel = runtime.GetObject<IPlantModel>();
       string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for specified
Node path
        IPlantObject plantObject = myPlantmodel.GetPlantObject(strNodeName);
        System.Console.WriteLine(string.Format("ViewName: {0} Name: {1}",
plantObject.CurrentPlantView,
        plantObject.Name));
        if (plantObject != null)
        {
            ICollection<string> PropNames = null;
            IPlantObjectPropertySet plantObjectPropertyset =
plantObject.GetProperties(PropNames);
            if (plantObjectPropertyset != null)
            {
                // Assign callback function
                plantObjectPropertyset.OnPlantModelPropertySubscriptionNotification +=
                odkPlantModelPropertySet_OnDataChanged;
                plantObjectPropertyset.Subscribe();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
public void odkPlantModelPropertySet_OnDataChanged(IPlantObjectPropertySet sender,
IList<IPlantObjectPropertyValue> Values)
{
    try
    {
        foreach (var value in Values)
        {
            Console.WriteLine("Name {0}", value.Name);
            Console.WriteLine("TimeStamp {0}", value.TimeStamp);
            Console.WriteLine("Value {0}", value.Value);
           Console.WriteLine("Quality {0}", value.Quality);Console.WriteLine("Error {0}",
value.Error);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
    finally
    {
        if (null != sender)
        {
            sender.Dispose();
        }
    }
```

Copy code

```
}
```

IPlantObjectAlarmSubscription

Description

The C# interface "IPlantObjectAlarmSubscription" specifies methods for monitoring alarms of "IPlantObject" instances.

The interface inherits the `Dispose()` method of the "IDisposable" interface of the .NET framework.

Member

The following properties, methods and events are specified in the interface:

"Filter" property

SQL-type string for filtering the result set of active alarms.

```
string Filter { get; set; }
```

All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 7828).

"Language" property

Country identifier of the language of the monitored alarms. See also section Locale IDs of the supported languages (Page 7829).

```
UInt32 Language { get; set; }
```

"IncludeChildren" property

If "true" is transferred, the alarm subscription only applies to the alarms of the "IPlantObject" instance and all its children in the hierarchy. If "false" is transferred, it only applies for the alarms of the "IPlantObject" instance.

```
bool IncludeChildren { get; set; }
```

"Start" method

Subscribe systems for monitoring changes of active alarms.

```
void Start()
```

"Stop" method

Clear monitoring of active alarms.

```
void Stop()
```

"OnPlantObjectSubscribeAlarmHandler" event

Declares the event for the monitoring of alarms of an "IPlantObject" instance.

The event calls an instance of the "OnPlantObjectSubscribeAlarmHandler" delegate.

```
event OnPlantObjectSubscribeAlarmHandler
OnPlantObjectSubscribeAlarmHandler;
```

"OnPlantObjectSubscribeAlarmHandler" delegate

Specifies the signature of the event handling method for the "OnPlantObjectSubscribeAlarmHandler" event of an "IPlantObject" instance.

```
public delegate void OnPlantObjectSubscribeAlarmHandler(
        IPlantObjectAlarmSubscription sender,
        UInt32 systemError,
        string systemName,
        IList<IAlarmResult> values);
```

- `sender`
  Source of the event
- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `values` is irrelevant.
- `systemName`
  Name of the runtime system that is subscribed for alarm monitoring by the user.
- `values`
  Event data as a list of "IAlarmResult" instances of the monitored active alarm.

Example

Copy code

```
public void Odk_GetAlarmSubscription()
{
    try
    {
        using (IPlantModel myPlantModel = runtime.GetObject<IPlantModel>())
        {
            string strNodeName = ".hierarchy::RootNodeName\\Node1";    //gets node for
specified Node path
            IPlantObject plantobject = myPlantModel.GetPlantObject(strNodeName);
           IPlantObjectAlarmSubscription alarmsub = plantobject.CreateAlarmSubscription();
  //Assign alarm handler
            if (alarmsub != null)
            {
                alarmsub.OnPlantObjectSubscribeAlarmHandler +=
alarm_OnPlantObjectSubscribeAlarmHandler;
                alarmsub.Filter = "";
                alarmsub.Language = 1033;
                alarmsub.IncludeChildren = false;
                alarmsub.Start();
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
public void alarm_OnPlantObjectSubscribeAlarmHandler(IPlantObjectAlarmSubscription sender,
UInt32 nGlobalError, String systemName, IList<IAlarmResult> value)
{
    try
    {
        foreach (var item in value)
        {
            System.Console.WriteLine(string.Format("Name: {0}", item.Name));
            System.Console.WriteLine(string.Format("InstanceID: {0}", item.InstanceID));
          System.Console.WriteLine(string.Format("AlarmClass: {0}", item.AlarmClassName));
            System.Console.WriteLine(string.Format("AlarmParameterValues: {0}",
item.AlarmParameterValues));
            System.Console.WriteLine(string.Format("AlarmText1: {0}", item.AlarmText1));
            System.Console.WriteLine(string.Format("Area: {0}", item.Area));
        }
    }
    finally
    {
        if (null != sender )
        {
            sender.Stop();
        }
    }
}
```

20.3.8.8 Interfaces of the Calendar option

ISHCCalendar

Description

The C# interface "ISHCCalendar" specifies the properties and methods of a calendar. The calendar is integrated via an "IPlantObject" instance.

The interface inherits the "Dispose() method of the "IDisposable" interface of the .NET framework and the methods of the "ISHCGetObject" interface

Members

"Settings" property

Reference to an "ISHCCalenderSettings" instance which saves the settings of the calendar.

```
ISHCCalendarSettings Settings { get; }
```

"Category" property

Reference to an "ISHCCategoryProvider" instance with which you access categories of the calendar.

```
ISHCCategoryProvider Category { get; }
```

"DayTemplate" property

Reference to an "ISHCDayTemplatesProvider" instance with which you create, read, update and delete day templates for the calendar.

```
ISHCDayTemplatesProvider DayTemplate { get; }
```

"ShiftTemplate" property

Reference to an "ISHCShiftTemplatesProvider" instance with which you create, read, update and delete shift templates for the calendar.

```
ISHCShiftTemplatesProvider ShiftTemplate { get; }
```

"ActionTemplate" property

Reference to an "ISHCActionTemplatesProvider" instance with which you create, read, update and delete action templates for the calendar.

```
ISHCActionTemplatesProvider ActionTemplate { get; }
```

"Day" property

Reference to an "ISHCDayProvider" instance with which you create, read, update and delete days for the calendar.

```
ISHCDayProvider Day { get; }
```

Example

The following example serves as a basis for the other examples for the C# interfaces of the Calendar option.

It shows how you can obtain the "IPlantObject" instance and also an "ISHCCalendar" instance. The "ISHCCalendar" instance referenced via `calendar` is also used in the other examples.

Copy code

```
using Siemens.Runtime.HmiUnified.SHC;
using Siemens.Runtime.HmiUnified;

ISHCCalendar calendar = null;
// Connect to Runtime
IRuntime runtime = Runtime.Connect();
if (runtime != null)
{
    IPlantModel myPlantModel = runtime.GetObject<IPlantModel>();
    IPlantObject po = myPlantModel.GetPlantObject(".hierarchy::Plant/Unit1");
    if (po != null)
    {
        calendar = po.Calendar();
    }
}
```

ISHCCategory

Description

The C# interface "ISHCCategory" specifies the properties and methods of a time category of the time model.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the category

```
string Name { get; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the category and its language code ID.

```
IDictionary<UInt32, string> Descriptions { get;  }
```

"Color" property

The color of the category

```
Color Color { get; }
```

"Deleted" property

Saves the information on whether an already used category was deleted in Engineering.

```
bool Deleted { get; }
```

Example

```
static void PrintCategory(ISHCCategory category)
{
     if (null != category)
     {
          Console.WriteLine(" \nName:{0} \nColor:{1} \n Deleted:{2}\n", category.Name,
category.Color, category.Deleted);
     }
     IDictionary<uint, string> displayNames = category.DisplayNames;
     foreach (var item in displayNames)
     {
          Console.WriteLine("Language:{0} DisplayName:{1}", item.Key, item.Value);
     }
     IDictionary<uint, string> description = category.Descriptions;
     foreach (var item in description)
     {
          Console.WriteLine("Language:{0} Description:{1}", item.Key, item.Value);
     }
}
```

See also

Locale IDs of the supported languages (Page 7829)

ISHCCategoryProvider

Description

The C# interface "ISHCCategoryProvider" provides you with read access to the "ISHCCategory" instances of an "ISHCCalendar" instance.

Members

"Browse" method

Supplies a collection with the categories of the calendar.

```
IReadOnlyCollection<ISHCCategory> Browse();
```

Example

Copy code

```
IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
foreach (var cat in categories)
{
    // do something
}
```

ISHCCalendarSettings

Description

The C# interface "ISHCCalendarSettings" specifies properties and methods for access to the calendar settings of an "ISHCCalendar" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"PlantObject" property

Reference to the "IPlantObject" instance to which the calendar belongs.

```
string PlantObject { get; }
```

"FirstDayOfWeek" property

Reference to the "ShcWeekDay" instance which is set as the first day of the week.

```
ShcWeekDay FirstDayOfWeek { get; }
```

"FirstWeekOfYear" property

Reference to the "ShcWeekStart" instance which is set as the first week of the year.

```
ShcWeekStart FirstWeekOfYear { get; }
```

"FiscalYearStartDay" property

The first day of the fiscal year

Default setting: 1

```
Byte FiscalYearStartDay { get; }
```

"FiscalYearStartMonth" property

The first month of the fiscal year

Default setting: 1

```
Byte FiscalYearStartMonth { get; }
```

"DayOffset" property

The offset with which the workday begins, calculated from midnight.

Default setting: 0

Maximum value: 24 hours

```
TimeSpan DayOffset { get; }
```

"Workdays" property

Number of workdays

```
UInt32 Workdays { get; }
```

"TimeZone" property

The Microsoft time zone

```
UInt32 TimeZone { get; }
```

Example

```
static void PrintCalendar(ISHCCalendarSettings calendar)
{
     if (null != calendar)
     {
          string cal = string.Format(" \n Workdays: {0} \n FirstDayOfWeek: {1} \n
FirstWeekOfYear: {2} \n FiscalYearStartDay: {3} \n FiscalYearStartMonth: {4} \n DayOffset:
{5} \n PlantObject: {6} \n \n TimeZone:{7} \n", calendar.Workdays, calendar.FirstDayOfWeek,
calendar.FirstWeekOfYear, calendar.FiscalYearStartDay, calendar.FiscalYearStartMonth,
calendar.DayOffset, calendar.PlantObject, calendar.TimeZone);
          Console.WriteLine(cal);
     }
}
```

ISHCTimeSlice

Description

The C# interface "ISHCTimeSlice" specifies the properties and methods of a time slice.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"StartTime" property

Time stamp with the start time of the time slice

```
DateTime StartTime { get; set; }
```

"Duration" property

The duration of the time slice

```
TimeSpan Duration { get; set; }
```

"Category" property

The time category of the time slice

```
string Category { get; set; }
```

ISHCDay

Description

The C# interface "ISHCDay" specifies the properties and methods of a day.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Comments" property

A dictionary from UInt32/string pairs with the comments of the "ISHCDay" instance and their language code IDs.

```
IDictionary<UInt32, string> Comments { get; }
```

"StartTime" property

Time stamp with the start time of the "ISHCDay" instance.

```
DateTime StartTime { get; set; }
```

"IsCustomized" property

Saves information on whether the "ISHCDay" instance was edited by users.

```
bool IsCustomized { get; }
```

"DayTemplate" property

The "ISHCDayTemplate" instance from which the "ISHCDay" instance is derived.

```
string DayTemplate { get; set; }
```

"CreateShift" method

Instantiates an "ISHCShift" instance at the "ISHCDay" instance.

```
ISHCShift CreateShift(
       ISHCShiftTemplate shcShiftTemplate,
       TimeSpan startTime);
```

- `shcShiftTemplate`
  Reference to the shift template from which the shift is derived
- `startTime`
  Time stamp with the start time of the "ISHCShift" instance.

"DeleteShift" method

Deletes a shift of the "ISHCDay" instance.

```
void DeleteShift(
       ISHCShift shcShift);
```

- `shcShift`
  Reference to the shift to be deleted

"GetShifts" method

Supplies a list with all the shifts of the "ISHCDay" instance.

```
IReadOnlyList<ISHCShift> GetShifts();
```

"SetComment" method

Adds a new entry to the dictionary of the "Comment" property.

```
void SetComment(
       UInt32 languageId,
       string comment);
```

- `languageId`
  The language code ID of the comment
- `comment`
  A comment

See also

Locale IDs of the supported languages (Page 7829)

ISHCDayProvider

Description

The C# interface "ISHCDayProvider" provides you with access to the days of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete days.

Members

"Browse" method

Supplies a collection with the "ISHCDay" instances of the calendar.

```
IReadOnlyCollection<ISHCDay> Browse(
       DateTime startTime, DateTime end);
```

- `startTime`
  Defines the start of the time period whose days are returned.
- `end`
  Defines the end of the time period whose days are returned.

Example:

```
static void ReadDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            foreach (var day in days)
            {
                // PrintDays(day);
                if (null != day)
               { string strDays = string.Format("\n StartTime :{0} \n IsCustomized :{1} \n
DayTemplate :{2} ", day.StartTime,                 day.IsCustomized, day.DayTemplate);
                Console.WriteLine(strDays);
                IDictionary<uint, string> Comments = day.Comments;foreach (var item in
Comments)
                {Console.WriteLine("\n Language:{0} DayComment:{1} \n", item.Key,
item.Value);  }
                }

            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Create" method

Adds new "ISHCDay" instances to the calendar.

```
void  Create(
       IList<ISHCDay> days);
```

- `days`
  List with the new "ISHCDay" instances

Example:

```
static void CreateDayWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> Daytemplates =
calendar.DayTemplate.Browse(false);
        if (Daytemplates.Count > 0)
        {
            ISHCDayTemplate dayTemplate = Daytemplates.ElementAt(0);
            List<ISHCDay> DayList = new List<ISHCDay>();
            ISHCDay day = calendar.GetObject<ISHCDay>();
            day.DayTemplate = dayTemplate.Name;
            DateTime dtday = DateTime.Now;
            day.StartTime = dtday;
            day.SetComment(1033, "DaywithShift");
            DayList.Add(day);
            calendar.Day.Create(DayList);
            IReadOnlyCollection<ISHCShiftTemplate> ShiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (ShiftTemplates.Count > 0)
            {
                ISHCShiftTemplate ShiftTemplate = ShiftTemplates.ElementAt(0);
              ISHCShift dayShift = day.CreateShift(ShiftTemplate, new TimeSpan(18, 0, 0));
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates "ISHCDay" instances of the calendar.

```
void Update(
        IList<ISHCDay> days);
```

- `days`
  List of the "ISHCDay" instances to be updated

Example:

```
static void UpdateDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(1);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            List<ISHCDay> list = new List<ISHCDay>();
            foreach (var day in days)
            {
                IReadOnlyCollection<ISHCShift> shifts = day.GetShifts();
                if (shifts != null)
                {
                    ISHCShift shift = shifts.ElementAt(0);
                    shift.GetTimeSlices().ElementAt(0).Category = "Maintenance";
                }
                list.Add(day);
            }
            calendar.Day.Update(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Delete" method

Deletes "ISHCDay" instances of the calendar.

```
void Delete(
      IList<ISHCDay> days);
```

- `days`
  List of "ISHCDay" instances to be deleted

Example:

```
static void DeleteDayWithShift()
{
    try
    {
        DateTime start = DateTime.Now.StartOfDay();
        DateTime end = DateTime.Now.EndOfDay();
        end = end.AddDays(3);
        IReadOnlyCollection<ISHCDay> days = calendar.Day.Browse(start, end);
        if (days.Count > 0)
        {
            List<ISHCDay> list = new List<ISHCDay>();
            foreach (var day in days)
            {
                list.Add(day);
            }
            calendar.Day.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

ISHCDayTemplate

Description

The C# interface "ISHCDayTemplate" specifies the properties and methods of a day.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCDayTemplate" instance.

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCDayTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCDayTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Deleted" property

Saves information on whether the day template was deleted by users.

```
bool Deleted { get; }
```

"SetDisplayName" method

Sets the display name of the "ISHCDayTemplate" instance and its language code ID.

```
void SetDisplayName(
       UInt32 languageId,
       string displayName);
```

- `languageId`
  The language code ID of the display name
- `displayName`
  The display name

"SetDescription" method

Sets the description of the "ISHCDayTemplate" instance and its language code ID.

```
void SetDescription(
       UInt32 languageId,
       string description);
```

- `languageId`
  The language code ID
- `description`
  The description

"GetShifts" method

Supplies a collection with the shifts of the "ISHCDayTemplate" instances.

```
IReadOnlyList<ISHCShift> GetShifts();
```

"CreateShift" method

Adds a shift to the "ISHCDayTemplate" instance.

```
ISHCShift CreateShift(
       ISHCShiftTemplate template,
       TimeSpan startTime);
```

- `template`
  Reference to the shift template on which the shift is based.
- `startTime`
  Time stamp with the start time of the shift

"DeleteShift" method

Deletes a shift of the "ISHCDayTemplate" instance.

```
void DeleteShift(
       ISHCShift shift);
```

- `shift`
  Reference to the shift to be deleted

See also

Locale IDs of the supported languages (Page 7829)

ISHCDayTemplatesProvider

Description

The C# interface "ISHCDayTemplatesProvider" provides you with access to the day templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete day templates.

Members

"Browse" method

Supplies a collection with the "ISHCDayTemplate" instances of the calendar.

```
IReadOnlyCollection<ISHCDayTemplate> Browse(
       bool includeDeleted);
```

- `includeDeleted`
  Saves information on whether the collection also contains the deleted day templates.

Example:

```
static void ReadDayTemplateWithShift()
{
    try
    {
         IReadOnlyCollection<ISHCDayTemplate> dayTemplate =
calendar.DayTemplate.Browse(false);
         if (dayTemplate.Count > 0)
             {
                 foreach (var template in dayTemplate)
                 {
                      PrintDayTemplates(template);
                      ReadShiftsforDayTemplate(template);
                 }
             }
     }
     catch (OdkException ex)
     {
         System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"Create" method

Adds new "ISHCDayTemplate" instances to the calendar.

```
void  Create(
       ICollection<ISHCDayTemplate> dayTemplates);
```

- dayTemplates
  Collection with the new "ISHCDayTemplate" instances

Example:

```
static void CreateDayTemplateWithShift()
{
    try
    {
        ISHCDayTemplate Daytemplate = calendar.GetObject<ISHCDayTemplate>();
        if (null != Daytemplate)
        {
            List<ISHCDayTemplate> ListDayTemplate = new List<ISHCDayTemplate>();
            Daytemplate.Name = "DayTemplateName";
            Daytemplate.SetDescription(1033, "DayTemplateDescription");
            Daytemplate.SetDisplayName(1033, "DayTemplateDisplayName");
            ListDayTemplate.Add(Daytemplate);
            calendar.DayTemplate.Create(ListDayTemplate);
            IReadOnlyCollection<ISHCShiftTemplate> ShiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (ShiftTemplates.Count > 0)
            {
                ISHCShiftTemplate ShiftTemplate = ShiftTemplates.ElementAt(0);
               ISHCShift shift = Daytemplate.CreateShift(ShiftTemplate, new TimeSpan(1, 0,
0));
            }
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Update" method

Updates the "ISHCDayTemplate" instances of the calendar.

```
void Update(
        ICollection<ISHCDayTemplate> dayTemplates);
```

- dayTemplates
  Collection with the "ISHCDayTemplate" instances to be updated

Example:

```
static void UpdateDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplate =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            List<ISHCDayTemplate> list = new List<ISHCDayTemplate>();
            foreach (var dayTemplates in dayTemplate)
            {
                dayTemplates.Name = "UpdatedDayTemplate";
                dayTemplates.SetDisplayName(1033, "UpdatedDayTemplateDisplayName");
                dayTemplates.SetDescription(1033, "UpdatedDayTemplateDescription");
                IReadOnlyCollection<ISHCShift> shifts = dayTemplates.GetShifts();
                if (shifts != null)
                {
                    ISHCShift shift = shifts.ElementAt(0);
                    shift.Duration = new TimeSpan(6, 0, 0);
                }
                list.Add(dayTemplates);
            }
            calendar.DayTemplate.Update(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

"Delete" method

Deletes "ISHCDayTemplate" instances of the calendar.

```
void Delete(
      ICollection<ISHCDayTemplate> dayTemplates);
```

- `dayTemplates`
  Collection with the "ISHCDayTemplate" instances to be deleted

Example:

```
static void DeleteDayTemplateWithShift()
{
    try
    {
        IReadOnlyCollection<ISHCDayTemplate> dayTemplates =
calendar.DayTemplate.Browse(false);
        if (dayTemplate.Count > 0)
        {
            List<ISHCDayTemplate> list = new List<ISHCDayTemplate>();
            foreach (var dayTemplate in dayTemplates)
            {
                list.Add(dayTemplate);
            }
            calendar.DayTemplate.Delete(list);
        }
    }
    catch (OdkException ex)
    {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
    }
}
```

ISHCShiftTemplate

Description

The C# interface "ISHCShiftTemplate" specifies the properties and methods of a shift.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCShiftTemplate" instance

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCShiftTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"Deleted" property

Saves information on whether the shift template was deleted by users.

```
bool Deleted { get; }
```

"Duration" property

The duration of the "ISHCShiftTemplate" instance.

```
TimeSpan Duration { get; set; }
```

"SetDisplayName" method

Sets the display name of the "ISHCShiftTemplate" instance and its language code ID.

```
void SetDisplayName(
       UInt32 languageId,
       string displayName);
```

- `languageId`
  The language code ID of the display name
- `displayName`
  The display name

"SetDescription" method

Sets the description of the "ISHCShiftTemplate" instance and its language code ID.

```
void SetDescription(
       UInt32 languageId,
       string description);
```

- `languageId`
  The language code ID
- `description`
  The description

"GetTimeSlices" method

Supplies a list with the time slices of the "ISHCShiftTemplate" instance.

```
IReadOnlyList<ISHCTimeSlice> GetTimeSlices();
```

"CreateTimeSlice" method

Adds a time slice to the "ISHCShiftTemplate" instance.

```
void CreateTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the new time slice

"DeleteTimeSlice" method

Deletes a time slice of the "ISHCShiftTemplate" instance.

```
void DeleteTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the time slice to be deleted

See also

Locale IDs of the supported languages (Page 7829)

ISHCShiftTemplatesProvider

Description

The C# interface "ISHCShiftTemplatesProvider" provides you with access to the shift templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete shift templates.

Members

"Browse" method

Supplies a collection with the "ISHCShiftTemplate" instances of the calendar.

```
IReadOnlyCollection<ISHCShiftTemplate> Browse(
       bool includeDeleted);
```

- `includeDeleted`
  Saves information on whether the collection also contains the deleted shift templates.

Example:

```
static void ReadShiftTemplatesWithTimeslice()
{
      try
      {
            Console.WriteLine("ReadShiftTemplate With Timeslice");
            IReadOnlyCollection<ISHCShiftTemplate> template =
calendar.ShiftTemplate.Browse(false);
            if (template.Count > 0)
            {
                  foreach (var shift in template)
                  {
                        PrintShiftTemplates(shift);
                        ReadTimeslicesforShiftTemplate(shift);
                  }
            }
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

"Create" method

Adds new "ISHCShiftTemplate" instances to the calendar.

```
void Create(
       ICollection<ISHCShiftTemplate> shiftTemplates);
```

- `shiftTemplates`
  Collection with the new "ISHCShiftTemplate" instances of the calendar.

Example:

```
static void CreateShiftTemplateWithTimeSlice()
{
      try
      {
         using (ISHCShiftTemplate pShiftTemplate = calendar.GetObject<ISHCShiftTemplate>())
            {
                pShiftTemplate.Name = "ShiftTemplateName";
                pShiftTemplate.SetDisplayName(1033, "ShiftTemplateDisplayName");
                pShiftTemplate.SetDescription(1033, "ShiftTemplateDescriptions");
                pShiftTemplate.Duration = new TimeSpan(8, 0, 0);
                List<ISHCShiftTemplate> ShiftList = new List<ISHCShiftTemplate>();
                ShiftList.Add(pShiftTemplate);
                calendar.ShiftTemplate.Create(ShiftList);
                IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
                if (categories.Count > 0)
                {
                    ISHCCategory pCat = categories.ElementAt(0);
                    ISHCTimeSlice pTimeSlice = calendar.GetObject<ISHCTimeSlice>();
                    pTimeSlice.StartTime = DateTime.Now.StartOfDay();
                    pTimeSlice.Duration = new TimeSpan(3, 0, 0);
                    pTimeSlice.Category = pCat.Name;
                    pShiftTemplate.CreateTimeSlice(pTimeSlice);
                }
            }
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

"Update" method

Updates "ISHCShiftTemplate" instances of the calendar.

```
void Update(
       ICollection<ISHCShiftTemplate> shiftTemplates);
```

- `shiftTemplates`
  Collection with the "ISHCShiftTemplate" instances to be updated

Example:

```
static void UpdateShiftTemplateWithTimeslice()
{
      try
      {
            IReadOnlyCollection<ISHCShiftTemplate> shiftTemplates =
calendar.ShiftTemplate.Browse(false);
            List<ISHCShiftTemplate> list = new List<ISHCShiftTemplate>();
            IReadOnlyCollection<ISHCCategory> categories = calendar.Category.Browse();
            ISHCCategory pCat = categories.ElementAt(1);
            foreach (var shifttemplate in shiftTemplates)
            {
                  shifttemplate.Name = "UpdateShiftTemplate";
                  shifttemplate.SetDisplayName(1033, "Updated DisplayName");
                  shifttemplate.SetDescription(1033, "UpdatedDescription");
                  shifttemplate.Duration = new TimeSpan(10, 0, 0);
                  list.Add(shifttemplate);
                  IReadOnlyCollection<ISHCTimeSlice> Timeslices =
shifttemplate.GetTimeSlices();
                  if (Timeslices.Count > 0)
                  {
                        ISHCTimeSlice timeslice = Timeslices.ElementAt(0);
                        timeslice.Duration = new TimeSpan(5, 0, 0);
                        timeslice.Category = pCat.Name;
                  }
            }
            calendar.ShiftTemplate.Update(list);
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

"Delete" method

Deletes "ISHCShiftTemplate" instances of the calendar.

```
void Delete(
       ICollection<ISHCShiftTemplate> shiftTemplates);
```

- `shiftTemplates`
  Collection with the "ISHCShiftTemplate" instances of the calendar to be deleted.

Example:

```
static void DeleteShiftTemplateWithTimeslice()
{
      try
      {
            IReadOnlyCollection<ISHCShiftTemplate> shiftTemplates =
calendar.ShiftTemplate.Browse(false);
            if (shiftTemplates.Count > 0)
            {
                  List<ISHCShiftTemplate> list = new List<ISHCShiftTemplate>();
                  foreach (var shifttemplate in shiftTemplates)
                  {
                        list.Add(shifttemplate);
                  }
                  calendar.ShiftTemplate.Delete(list);
            }
      }
      catch (OdkException ex)
      {
            System.Console.WriteLine(string.Format("OdkException occurred {0}",
ex.Message));
      }
}
```

ISHCShift

Description

The C# interface "ISHCShift" specifies the properties and methods of a shift.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"StartTime" property

Time stamp with the start time of the "ISHCShift" instance.

```
DateTime StartTime { get; set; }
```

"Duration" property

The duration of the "ISHCShift" instance.

```
TimeSpan Duration { get; set; }
```

"ShiftTemplate" property

The shift template of the "ISHCShift" instance.

```
string ShiftTemplate { get; }
```

"IsCustomized" property

Saves information on whether the "ISHCShift" instance was edited by users.

```
bool IsCustomized { get; }
```

"DeltaKind" property

Saves information on how the time slices of the "ISHCShift" instance deviate from the shift template.

```
ShcDeltaType DeltaKind { get; }
```

The enumeration "ShcDeltaType" can contain the following values:

- `Added` (0)
- `Modified` (1)
- `Deleted` (2)

"ShiftId" property

Saves the ShiftID of the "ISHCShift" instance.

```
UInt32 ShiftId { get; set; }
```

"Comments" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCShift" instance and their language code IDs.

```
IDictionary<UInt32, string> Comments { get; }
```

"GetTimeSlices" method

Supplies a collection with the time slices of the "ISHCShift" instances.

```
IReadOnlyList<ISHCTimeSlice> GetTimeSlices();
```

"CreateTimeSlice" method

Adds a time slice to the "ISHCShift" instance.

```
void CreateTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the new time slice

"DeleteTimeSlice" method

Deletes a time slice of the "ISHCShift" instance.

```
void DeleteTimeSlice(
       ISHCTimeSlice slice);
```

- `slice`
  Reference to the time slice to be deleted

"SetComment" method

Adds a comment with language code ID to the "Comments" property.

```
void SetComment(
       UInt32 languageId,
       string comment);
```

- `languageId`
  The language code ID of the comment
- `comment`
  The comment

"CreateAction" method

Adds an action to the "ISHCShift" instance.

```
ISHCAction CreateAction(
       ISHCActionTemplate actionTemplate,
       TimeSpan offset);
```

- `actionTemplate`
  The action template of the new action
- `offset`
  The offset for the anchor point of the action, in relation to the starting point of the shift. Positive and negative value allowed.

Example:

```
static void CreateActionUsingShift()
{
     try
     {
          DateTime start = DateTime.Now.StartOfDay();
          DateTime end = DateTime.Now.EndOfDay();
          end = end.AddDays(3);
          ISHCShift Shift = calendar.Day.Read(start,
end).ElementAt(0).GetShifts().ElementAt(0);
          if (Shift != null)
          {
               ISHCAction Action =
Shift.CreateAction(calendar.ActionTemplate.Read(false).ElementAt(0), new TimeSpan(5, 0,
0));
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"DeleteAction" method

Deletes an action of the "ISHCShift" instance.

```
void DeleteAction(ISHCAction shcAction);
```

- `shcAction`
  Reference to the action to be deleted

"GetActions" method

Supplies a list with the actions of the "ISHCShift" instance.

```
IReadOnlyList<ISHCAction> GetActions();
```

Example:

```
static void ReadActionUsingShift()
{
     try
     {
          DateTime start = DateTime.Now.StartOfDay();
          DateTime end = DateTime.Now.EndOfDay();
          end = end.AddDays(3);
          ISHCShift Shift = calendar.Day.Read(start,
end).ElementAt(0).GetShifts().ElementAt(0);
          if (Shift != null)
          {
               IReadOnlyList<ISHCAction> action = Shift.GetActions();
               if (action != null)
               {
                    ISHCAction actions = action.ElementAt(0);
                    string Action = string.Format("\n Offset:{0} \n IsCustomized:
{1} ,actionTemplate:{2}", actions.Offset, actions.IsCustomized, actions.ActionTemplate);
                    System.Console.WriteLine(Action);
               }
          }
     }
     catch (OdkException ex)
     {
         System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

See also

Locale IDs of the supported languages (Page 7829)

ISHCAction

Description

The C# interface "ISHCAction" specifies the properties and methods of an action.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Offset" property

The offset for the anchor point of the "ISHCAction" instance in 100 nanoseconds in relation to the start point of its shift instance. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"ActionTemplate" property

The action template of the "ISHCAction" instance

```
string ActionTemplate { get; }
```

"IsCustomized" property

Saves information on whether the "ISHCAction" instance was edited by users.

```
bool IsCustomized { get; }
```

"GetElements" method

Supplies a list with the action elements of the "ISHCAction" instance.

```
IReadOnlyList<ISHCActionElement> GetElements();
```

ISHCActionElement

Description

The C# interface "ISHCActionElement" specifies the properties and methods of an action element of an "ISHCAction" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"ElementType" property

The type of the "ISHCActionElement" instance.

```
ShcActionElementType ElementType { get; }
```

The enumeration "ShcActionElementType" can contain the following values:

- `Tag` (0)
  The action element controls a tag.

"Enabled" property

Saves the information on whether the "ISHCActionElement" instance is activated.

```
bool Enabled { get; set; }
```

"Offset" property

The offset of the "ISHCActionElement" instance in 100 nanoseconds in relation to the anchor point of its action. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"Value" property

Value of the tag controlled by the "ISHCActionElement" instance

```
object Value { get; set; }
```

"ElementName" property

Name of the tag controlled by the "ISHCActionElement" instance

```
string ElementName { get; set; }
```

ISHCActionTemplate

Description

The C# interface "ISHCActionTemplate" specifies the properties and methods of the action template of an "ISHCAction" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"Name" property

The name of the "ISHCActionTemplate" instance.

```
string Name { get; set; }
```

"DisplayNames" property

A dictionary from UInt32/string pairs with the display names of the "ISHCActionTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> DisplayNames { get; }
```

"Deleted" property

Saves information on whether the action template was deleted by users.

```
bool Deleted { get; }
```

"Descriptions" property

A dictionary from UInt32/string pairs with the descriptions of the "ISHCActionTemplate" instance and their language code IDs.

```
IDictionary<UInt32, string> Descriptions { get; }
```

"SetDisplayName" method

Sets the display name of the "ISHCActionTemplate" instance and its language code ID.

```
void SetDisplayName(
       UInt32 languageId,
       string displayName);
```

- `languageId`
  The language code ID of the display name
- `displayName`
  The display name

"SetDescription" property

Sets the description of the "ISHCActionTemplate" instance and its language code ID.

```
void SetDescription(
       UInt32 languageId,
       string description);
```

- `languageId`
  The language code
- `IDdescription`
  The description

"CreateElement" property

Adds an "ISHCActionTemplateElement" instance to the "ISHCActionTemplate" instance.

```
void CreateElement(
       ISHCActionTemplateElement actiontemplateElement);
```

- `actiontemplateElement`
  Reference to the new action template element

"DeleteElement" property

Deletes an "ISHCActionTemplateElement" instance of the "ISHCActionTemplate" instance.

```
void DeleteElement(
       ISHCActionTemplateElement actiontemplateElement);
```

- `actiontemplateElement`
  Reference to the action template element to be deleted

"GetElements" property

Supplies a list with action template elements of the "ISHCActionTemplate" instances.

```
IReadOnlyList<ISHCActionTemplateElement> GetElements();
```

See also

Locale IDs of the supported languages (Page 7829)

ISHCActionTemplateElement

Description

The C# interface "ISHCActionTemplateElement" specifies the properties and methods of an action element of an "ISHCActionTemplate" instance.

The interface inherits the "Dispose()" method of the "IDisposable" interface of the .NET framework.

Members

"ElementType" property

The type of the "ISHCActionTemplateElement" instance.

```
ShcActionElementType ElementType { get; }
```

The enumeration "ShcActionElementType" can contain the following values:

- `Tag` (0)
  The action element controls a tag.

"Offset" property

The offset of the "ISHCActionTemplateElement" instance in 100 nanoseconds in relation to the anchor point of its action template. Positive and negative value allowed.

```
TimeSpan Offset { get; set; }
```

"Value" property

Value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
object Value { get; set; }
```

"ElementName" property

Name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
string ElementName { get; set; }
```

ISHCActionTemplatesProvider

Description

The C# interface "ISHCActionTemplatesProvider" provides you with access to the action templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete action templates.

Members

"Browse" method

Supplies a collection with the "ISHCActionTemplate" instances of the calendar.

```
IReadOnlyCollection<ISHCActionTemplate> Browse(
       bool includeDeleted);
```

- `includeDeleted`
  Saves information on whether the collection also contains the deleted action templates.

Example:

```
static void ReadActionTemplate()
{
     try
     {
          Console.WriteLine("ReadActionTemplate");
          IReadOnlyCollection<ISHCActionTemplate> actionTemplate =
calendar.ActionTemplate.Browse(false);
          foreach (var template in actionTemplate)
          {
               PrintActionTemplates(template);
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"Create" method

Adds new "ISHCActionTemplate" instances to the calendar.

```
void  Create(
       ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
  Collection with the new "ISHCActionTemplate" instances

Example:

```
static void CreateActionTemplateWithActionTemplateElement()
{
     try
     {
          ISHCActionTemplate pActionTemplate = calendar.GetObject<ISHCActionTemplate>();
          if (pActionTemplate != null)
          {
               pActionTemplate.Name = "ActionTemplate";
               pActionTemplate.SetDisplayName(1033, "ActionDisplayName");
               pActionTemplate.SetDescription(1033, "ActionDescription");
               List<ISHCActionTemplate> ActionList = new List<ISHCActionTemplate>();
               ActionList.Add(pActionTemplate);
               calendar.ActionTemplate.Create(ActionList);
               ISHCActionTemplateElement pActionTemplateElement =
calendar.GetObject<ISHCActionTemplateElement>();
               if (pActionTemplateElement != null)
               {
                    pActionTemplateElement.ElementName = "HMI_RT_1::Unit1.Member_1";
                    pActionTemplateElement.Value = false;
                    pActionTemplateElement.Offset = new TimeSpan(4, 0, 0);
                    pActionTemplate.CreateElement(pActionTemplateElement);
               }
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

"Update" method

Updates "ISHCActionTemplate" instances of the calendar.

```
void Update(
        ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
  Collection with the "ISHCActionTemplate" instances to be updated

Example:

```
static void UpdateActionTemplateWithActionTemplateElement()
{
     IReadOnlyCollection<ISHCActionTemplate> actionTemplates =
calendar.ActionTemplate.Browse(false);
     List<ISHCActionTemplate> list = new List<ISHCActionTemplate>();
     foreach (var actionTemplate in actionTemplates)
     {
          actionTemplate.Name = "UpdatedActionTemplate";
          actionTemplate.SetDisplayName(1033, "UpdatedDisplayName ");
          actionTemplate.SetDescription(1033, "UpdatedDescription");
          IReadOnlyCollection<ISHCActionTemplateElement> action =
actionTemplate.GetElements();
          ISHCActionTemplateElement templateElement = action.ElementAt(0);
          templateElement.Offset = new TimeSpan(6, 0, 0);
          list.Add(actionTemplate);
     }
     calendar.ActionTemplate.Update(list);
}
```

"Delete" method

Deletes "ISHCActionTemplate" instances of the calendar.

```
void Delete(
      ICollection<ISHCActionTemplate> actionTemplates);
```

- `actionTemplates`
  Collection with the "ISHCActionTemplate" instances to be deleted

Example:

```
static void DeleteActionTemplateWithActionTemplateElement()
{
     try
     {
          IReadOnlyCollection<ISHCActionTemplate> actionTemplates =
calendar.ActionTemplate.Browse(false);
          if (actionTemplates.Count > 0)
          {
               List<ISHCActionTemplate> list = new List<ISHCActionTemplate>();
               foreach (var actionTemplate in actionTemplates)
               {
                    list.Add(actionTemplate);
               }
               calendar.ActionTemplate.Delete(list);
          }
     }
     catch (OdkException ex)
     {
        System.Console.WriteLine(string.Format("OdkException occurred {0}", ex.Message));
     }
}
```

20.3.8.9 Interfaces of the contexts

IContextLogging

Description

The C# interface "IContextLogging" defines events and methods for creating and reading "IContextDefinition" instances as well as for starting, stopping, monitoring and reading their "ILoggedContext" instances. You can use "ILoggedContext" instances to filter runtime data, for example, for alarms that fall within the time period of a particular "ILoggedContext" instance.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

The methods trigger an exception in case of an error.

Members

"CreateContextDefinitions" method

Creates ContextDefinitions in the database.

```
void CreateContextDefinitions(ICollection<IContextDefinition>
contextDefinitions)
```

- `contextDefinitions`:
  Collection with "IContextDefinition" instances

"ReadContextDefinitions" method

Reads ContextDefinitions from the database. The instances can be filtered by plant object and HMIContextProviderType .

```
void ReadContextDefinitions(ICollection<string> plantViewPaths =
null, ICollection<HmiContextProviderType> providerTypes = null,
HmiSortingMode sortingMode = HmiSortingMode.Ascending)
```

- `plantViewPaths`:
  Limits the read operation to "IContextDefinition" instances from this collection of plant objects.
- (optional) `providerTypes`:
  Limits the read operation to "IContextDefinition" instances with HmiContextProvider types from this collection.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- (optional) `sortingMode`:
  The "HmiSortingMode" according to which the read "IContextDefinition" instances are sorted.
  The enumeration "HmiSortingMode" can contain the following values:
  - Ascending = 1
    Default setting
  - Descending = 2

"ReadContexts" method

Reads "ILoggedContext" instances of a specific time period. The instances can be filtered using a "IContextFilter" instance.

```
void ReadContexts(DateTime start, DateTime end,
IContextFilter ContextFilter=null, HmiSortingMode sortingMode =
HmiSortingMode.Ascending)
```

- `start`:
  The start time of the period within which the "ILoggedContext" instances must lie.
- `end`:
  The end time of the period within which the "ILoggedContext" instances must lie.

- (optional) `ContextFilter`:
  The "IContextFilter" instance whose filter settings are used.
- (optional) `sortingMode`:
  The "HmiSortingMode" according to which the read "ILoggedContext" instances are sorted.
  The enumeration "HmiSortingMode" can contain the following values:
  - Ascending = 1
    Default setting
  - Descending = 2

"StartContext" method

Creates a new "ILoggedContext" instance for a "IContextDefinition" instance.

```
void StartContext(string contextName, HmiContextProviderType
providerType, string plantViewPath, object contextValue, DateTime
startTime, UInt32 qualityCode)
```

- `contextName`:
  The name of the "IContextDefinition" instance for which the context log entry is created.
- (optional) `providerType`:
  The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `plantViewPath`:
  Path to an "IPlantObject" instance
  E.g.: ".hierarchy::PlantView/Ln_1/M_1"
  Is used in combination with `contextName` for unique identification of the "IContextDefinition" instance.
- `contextValue`:
  The context value of the context log entry
- `startTime`:
  The start time of the new context log entry
- `qualityCode`:
  The QualityCode of the context value of the context log entry

"StopContext" method

Stops the currently running "ILoggedContext" instance of an "IContextDefinition" instance.

```
void StopContext(string contextName, HmiContextProviderType
providerType, string plantViewPath, DateTime endtime)
```

- `contextName`:
  The name of the "IContextDefinition" instance whose context log entry is stopped.
- (optional) `providerType`:
  The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `plantViewPath`:
  Path to an "IPlantObject" instance
  E.g.: ".hierarchy::PlantView/Ln_1/M_1"
  Is used in combination with `contextName` for unique identification of the "IContextDefinition" instance.
- `endtime`:
  End time of the context log entry

"Add" method

Adds a "IContextDefintion" instance to a vector. The methods Clear(), Subscribe() and CancelSubscription() can be called for the instances of the vector.

```
void Add(string contextName, HmiContextProviderType providerType,
string plantViewPath)
```

- `contextName`:
  The name of the "IContextDefintion" instance
- `providerType`:
  The HmiContextProviderType of the "IContextDefintion" instance
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `plantViewPath`:
  Path to an "IPlantObject" instance
  E.g.: ".hierarchy::PlantView/Ln_1/M_1"
  Is used in combination with `contextName` for unique identification of the "IContextDefinition" instance.

"Clear" method

Deletes the "IContextDefintion" instances added via `Add()` from the vector.

```
void Clear()
```

"Subscribe" method

Subscribes the "IContextDefinition" instances added to the vector via `Add()` for monitoring.

```
void Subscribe()
```

"CancelSubscription" method

Unsubscribes the "IContextDefinition" instances added to the vector with `Add()` from monitoring.

```
void CancelSubscribe()
```

"OnContextDefinitionCreate" event

The event calls the delegate "OnContextDefinitionCreateDelegate" after the creation of ContextDefinitions.

Declares the event and the event handler for creating "IContextDefinition" instances.

```
event OnContextDefinitionCreateDelegate OnContextDefinitionCreate;
```

"OnContextDefinitionReadReply" event

After reading the ContextDefinitions, the event calls the "OnContextDefinitionReadReplyDelegate" delegate.

Declares the event and the event handler for reading "IContextDefinition" instances.

```
event OnContextDefinitionReadReplyDelegate
OnContextDefinitionReadReply;
```

"OnLoggedContextReadReply" event

After reading the LoggedContexts, the event calls the "OnContextReadReplyDelegate" delegate.

Declares the event and the event handler for reading "ILoggedContext" instances.

```
event OnContextReadReplyDelegate OnLoggedContextReadReply;
```

"OnContextDataChanged" event

Event calls the delegate "OnContexDataChangedHandler" when starting or stopping a monitored "ILoggedContext" instance.

Declares the event and the event handler for monitoring "ILoggedContext" instances.

```
event OnContexDataChangedHandler OnContextDataChanged;
```

"OnContextDefinitionCreateDelegate" delegate

Specifies the signature of the event handling method for the "OnContextDefinitionCreate" event of the "IContextLogging" interface.

```
public delegate void
OnContextDefinitionCreateDelegate(IContextLogging sender,
        UInt32 globalError,
        string systemName,
        List<IContextError> errors,
        bool completed)
```

- `sender`:
  Source of the event
- `globalError`:
  Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- `systemName`:
  Name of the system on which the ContextDefinitions have been created.
- `errors`:
  List with the instance-specific errors that were generated when the ContextDefinitions were created.
- `completed`:
  Status of the asynchronous transfer:
  - True: All ContextDefinitions have been notified.
  - False: Additional notifications are expected.

"OnContextDefinitionReadReplyDelegate" delegate

Specifies the signature of the event handling method for the "OnContextDefinitionReadReply" event of the "IContextLogging" interface.

```
public delegate void
OnContextDefinitionReadReplyDelegate(IContextLogging sender,
        UInt32 globalError,
        string systemName,
        IList<IContextDefinition> contextDefinitionData,
        bool completed)
```

- `sender`:
  Source of the event
- `globalError`:
  Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- `systemName`:
  Name of the system on which the ContextDefinitions have been created.
- `contextDefinitionData`:
  List of read "IContextDefinition" instances
- `completed`:
  Status of the asynchronous transfer:
  - True: All ContextDefinitions have been notified.
  - False: Additional notifications are expected.

"OnContextReadReplyDelegate" delegate

Specifies the signature of the event handling method for the "OnLoggedContextReadReply" event of the "IContextLogging" interface.

```
public delegate void OnContextReadReplyDelegate(IContextLogging
sender,
        UInt32 globalError,
        string systemName,
        IList<ILoggedContext> loggedContexts,
        bool completed)
```

- `sender`:
  Source of the event
- `globalError`:
  Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- `systemName`:
  Name of the system on which the "ILoggedContext" instances have been created.
- `loggedContexts`:
  List with "ILoggedContext" instances that were started or stopped.
- `completed`:
  Status of the asynchronous transfer:
  - True: All ContextDefinitions have been notified.
  - False: Additional notifications are expected.

"OnContexDataChangedHandler" delegate

Specifies the signature of the event handling method for the "OnContextDataChanged" event of the "IContextLogging" interface.

```
public delegate void OnContexDataChangedHandler(IContextLogging
sender,
        IList<ILoggedContext> loggedContexts)
```

- `sender`:
  Source of the event
- `loggedContexts`:
  List with "ILoggedContext" instances

Example

Copying code

```
public void CreateContextDefinitions()
{
     Console.WriteLine("CreateContextDefinitions \n");
     var contextLogging = _runtime.GetObject<IContextLogging>();
     List<IContextDefinition> contextDefnitions = new List<IContextDefinition>();
     for (int i = 0; i < 5; i++)
     {
          var contextDefinition = _runtime.GetObject<IContextDefinition>();
          contextDefinition.PlantViewPath = ".hierarchy::Plant/Node1_1";
          contextDefinition.DataType = HmiContextDataType.DInt;
          Dictionary<UInt32, string> displayLanguages = new Dictionary<uint, string>();
          displayLanguages[1033] = "english";
          displayLanguages[1031] = "deutsch";
          contextDefinition.DisplayNames = displayLanguages;
          Random rnd = new Random();
          int num = rnd.Next(1, 1000);
          contextDefinition.Name = "CD_" + num + i.ToString();
          contextDefnitions.Add(contextDefinition);
          contextLogging.OnContextDefinitionCreate +=
ContextLogging_OnContextDefinitionCreate;
          contextLogging.CreateContextDefinitions(contextDefnitions);
          _event.WaitOne();
          _event.Reset();
          contextLogging.Dispose();
     }
}
```

Copying code

```
private void ContextLogging_OnContextDefinitionCreate(IContextLogging sender, UInt32
globalError, string systemName, IList<IContextError> errors, bool completed)
{
    if (globalError != 0)
    {
        Console.WriteLine("System Name:{0} Error:{1}", systemName, globalError);
    }
    else
    {
        if (null != errors)
        {
            foreach (var error in errors)
            {
                Console.WriteLine("ContextName:{0} Error:{1}", error.Name, error.Error);
                Console.WriteLine();
            }
        }
    }
}
```

Copying code

```
public void ReadContextDefinitionsWithFilter()
{
    Console.WriteLine("ReadContextDefinitionsWithFilter \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    contextLogging.OnContextDefinitionReadReply +=
OnContextDefinitionReadReplyForContextLogging;
    List<string> plantobjectsfilter = new List<string>();
    plantobjectsfilter.Add(".hierarchy::Plant/Node1_1");
    List<HmiContextProviderType> contextProviderType = new List<HmiContextProviderType>();
    contextProviderType.Add(HmiContextProviderType.UserDefined);
    contextLogging.ReadContextDefinitions(plantViewPaths: plantobjectsfilter,
providerTypes: contextProviderType, sortingMode: HmiSortingMode.Ascending);
    _event.WaitOne();
    _event.Reset();
    contextLogging.Dispose();
}
```

Copying code

```
public void StartContext()
{
    Console.WriteLine("StartContext \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    startDt = System.DateTime.Now;
    object value = "Orange Juice";
    uint quality = 192;
    var strContextName = "ContextName_" + strContext_Unique_Num;
    var providerType = HmiContextProviderType.UserDefined;
    IPlantObject plantObject = ".hierarchy::PlantView/Ln_1/M_1";
    var plantViewPath = ".hierarchy::Plant/Node1_1";
    contextLogging.StartContext(strContextName, providerType, plantViewPath, value,
startDt, quality);
}
```

Copying code

```
public void StopContext()
{
    Console.WriteLine("StopContext \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    endDt = System.DateTime.Now;
    var strContextName = "ContextName_" + strContext_Unique_Num;
    var providerType = HmiContextProviderType.UserDefined;
    contextLogging.StopContext(strContextName, providerType, endDt);
}
```

Copying code

```
void OnContextDefinitionReadReplyForContextLogging(IContextLogging sender, UInt32
globalErrors, string systemName, IList<IContextDefinition> contextDefinitions, bool
completed)
{
    if (globalErrors != 0)
    {
     Console.WriteLine("System Name:{0} Error:{1}", systemName, globalErrors);
    }
    else
    {
        if (null != contextDefinitionData)
        {
            foreach (var cd in contextDefinitionData)
            {
                Console.WriteLine("PlantViewPath:{0} Name:{1} Datatype:{2} Error:{3}",
cd.PlantViewPath, cd.Name, cd.DataType, cd.Error);
                Console.WriteLine();
            }
        }
    }
}
```

Copying code

```
void OnLoggedContextReadReplyForContextLogging(IContextLogging sender, UInt32
globalErrors, string systemName, IList<ILoggedContext> loggedContexts, bool completed)
{
    if (globalErrors != 0)
    {
        Console.WriteLine("System Name:{0} Error:{1}", systemName, globalErrors);
    }
    else
    {
        if (null != loggedContexts)
        {
            foreach (var lc in loggedContexts)
            {
               Console.WriteLine("StartTime:{0} EndTime:{1} Value:{2} Quality:{3} Error:
{4}", lc.StartTime, lc.EndTime, lc.Value, lc.Quality, lc.Error);
               Console.WriteLine();
            }
        }
    }
}
```

Copying code

```
public void Subscribe()
{
    Console.WriteLine("Subscribe \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    contextLogging.OnContextDataChanged += OnContextDataChangedForContextLogging;
    CreateContextDefinition();
    var name = "ContextName_" + strContext_Unique_Num;
    var providerType = HmiContextProviderType.UserDefined;
    contextLogging.Add(name, providerType);
    contextLogging.Subscribe();
    StartContext();
    StopContext();
    contextLogging.CancelSubscribe();
    contextLogging.Dispose();
}
```

IContextDefinition

Description

The C# interface "IContextDefinition" specifies properties for the definition of "IContextDefinition" instances. "ILoggingContext" instances can be created using the Start() and Stop() methods based on an "IContextDefiniton" instance.

The interface inherits the Dispose() method of the "IDisposable" interface of the .NET framework.

Members

"PlantViewPath" property

Sets the path to the plant object of the "IContextDefinition" instance

```
string PlantViewPath { get; set; }
```

Example: ".hierarchy::Plant/Station"

"ProviderType" property

The source that creates the instance.

Is used together with "Name" to uniquely identify a "IContextDefinition" instance.

```
HmiContextProviderType ProviderType { get; }
```

The enumeration "HmiContextProviderType" can contain the following values:

(optional) `providerType`:
The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.

The enumeration "HmiContextProviderType" can contain the following values:

- NoContext = 0
- Calendar = 1
  For "IContextDefinition" instances generated by the PI Option Calendar.
- PerformanceInsightMachineState = 2
  For "IContextDefinition" instances generated by the PI Option Performance Insight.
- LineCoordinator = 3
  For "IContextDefinition" instances generated by the PI Option Line Coordinator.
- UserDefined = 5
  It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"Name" property

The name of the "IContextDefinition" instance

Is used together with "ProviderType" to uniquely identify a "IContextDefinition" instance.

```
string Name { get; set;}
```

"DisplayNames" property

The display name of the "IContextDefinition" instance

```
IDictionary<UInt32, string> DisplayNames { get; set; }
```

"DataType" property

The data type of the "IContextDefinition" instance

```
HmiContextDataType DataType { get; set; }
```

The enumeration "HmiContextDataType" can contain the following values:

- Bool = 0x01
- SInt = 0x02

- Int = 0x03
- DInt = 0x04
- LInt = 0x05
- USInt = 0x06
- UInt = 0x07
- UDInt = 0x08
- ULInt = 0x09
- Real = 0x0A
- LReal = 0x0B
- LTime = 0x0C
- DateTime = 0x0D
- Byte = 0x11
- Word = 0x12
- DWord = 0x13
- LWord = 0x14
- String = 0x32

"Error" property

The error code of the "IContextDefinition" instance

Is set if the instance is read incorrectly.

```
UInt32 Error { get; }
```

ILoggedContext

Description

The C# interface "ILoggedContext" defines properties of context log entries of an "IContextDefinition" instance.

The context log entries are started and stopped using methods of the "IContextLogging" interface.

Members

"StartTime" property

The start time of the "IContextLogging" instance

```
DateTime StartTime { get; }
```

"EndTime" property

The end time of the "IContextLogging" instance

```
DateTime EndTime { get; }
```

"Error" property

The error code of the "IContextLogging" instance

```
UInt32 Error { get; }
```

"Value" property

The value of the "IContextLogging" instance

The value has the same data type as is specified by the "DataType" property of the "IContextDefinition" instance.

Example: An "IContextDefinition" instance has the name "Product" and the data type String. Its "IContextLogging" instance has the value "Limo".

```
object Value { get; }
```

"Quality" property

The QualityCode of the context value

```
UInt32 Quality { get; }
```

IContextError

Description

The C# interface "IContextError" specifies properties of error results that occur when generating ContextDefinitions in the database.

Members

"Name" property

Name of the "IContextDefintion" instance

```
string Name { get; }
```

"Error" property

The error code

```
UInt32 Error { get; }
```

IContextFilter

Description

The C# interface "IContextFilter" specifies the properties for filtering according to "ILoggedContext" instances.

Members

"Name" property

The name of the "IContextDefinition" instance for whose "ILoggedContext" instances filtering is performed.

```
string Name { set; get; }
```

"ProviderType" property

The HmiContextProviderType of an "IContextDefinition" instance for whose "ILoggedContext" instances the filtering is performed.

```
HmiContextProviderType ProviderType { set; get;}
```

The enumeration "HmiContextProviderType" can contain the following values:

- NoContext = 0
- Calendar = 1
  For "IContextDefinition" instances generated by the PI Option Calendar.
- PerformanceInsightMachineState = 2
  For "IContextDefinition" instances generated by the PI Option Performance Insight.
- LineCoordinator = 3
  For "IContextDefinition" instances generated by the PI Option Line Coordinator.
- UserDefined = 5
  It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"Operator" property

The filter operator

```
string Operator { set; get; }
```

The operator is applied to the value. The following operators are allowed:

- For values with data type Int and Real:
  - =
  - !=
  - <
  - >
  - <=
  - >=
- For values with data type String:
  - LIKE
  - =

  Strings must always be fully specified.

```
string Operator { set; get; }
```

Examples of operators: ">", "<", ">=" , "<=" and "="

"Value" property

To filter by "Value" of the "ILoggedContext" instance

```
object Value { set; get; }
```

Example

Copying code

```
public void ReadContextWithFilter()
{
    Console.WriteLine("ReadContextWithFilter \n");
    var contextLogging = _runtime.GetObject<IContextLogging>();
    contextLogging.OnLoggedContextReadReply += OnLoggedContextReadReplyForContextLogging;
    IContextFilter Filter = _runtime.GetObject<IContextFilter>();
    if (Filter != null)
    {
     Filter.Name = "ContextName_" + strContext_Unique_Num;
     Filter.ProviderType = HmiContextProviderType.UserDefined;
     Filter.Operator = "=";
     Filter.Value = "Orange Juice";
    }
    contextLogging.ReadContexts(startDt, endDt, Filter, HmiSortingMode.Ascending);
    _event.WaitOne();
    _event.Reset();
    contextLogging.Dispose();
}
```

20.3.9 Description of the C++ interfaces

20.3.9.1 Error codes of the C++ interfaces

All methods that have defined a `CFRESULT` return `CFSUCCESS` if the method was run through successfully. Otherwise, they return a corresponding error code.

20.3.9.2 Interfaces of the Runtime environment

IOdkRt

Description

The C++ interface "IOdkRt" specifies methods for the connection to the Runtime system and the error handling.

Members

The following methods are specified in the interface:

"Connect" method

Checks whether valid licenses are available for the products installed on the Runtime device:

- Yes: Connects ODK application and Runtime project.
- No: Supplies an error code. Use the interfaces for error handling to query the error description (license missing, expired, etc.).

```
CFRESULT Connect(
         const CFSTR context,
         IRuntime **ppRuntime,
         const CTSTR user = nullptr,
         const CTSTR password = nullptr)
```

- `context`
  [in]: Name of the runtime project

  Note

  The name of the Runtime project is not used in the current version. An empty string must be passed in order to connect to the locally run Runtime project.

- `IRuntime`
  [out]: Points to the initialized "IRuntime" object that the ODK object model makes available.
- `user`
  [in]: User name

  Note

  Can only be used in a future version!

- `password`
  [in]: Password

  Note

  Can only be used in a future version!

"Close" method

Enable configuration files and plug-ins of the Runtime system.

```
CFRESULT Close()
```

"GetErrorHandler" method

Transfers an "IErrorInfo" object for error handling.

```
CFRESULT GetErrorHandler(IErrorInfo** pErrorInfo)
```

`IErrorInfo`
[out]: Points to an "IErrorInfo" object.

Example

Connect to the Runtime system of the active project:

Copy code

```
IRuntimePtr pRuntime;

CFRESULT Connect()
{
    // Connect to running project
    CCfString projectName = L"";
    CFRESULT retVal = Connect(projectName, &pRuntime);

    return retVal;
}
```

Error handling when reading out installed options of the Runtime system:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    //load option component by name
    pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnknown;
    //create a instance of the option object "MyOptionObject"
    pOdkOption->GetObject(CCfString("MyOptionObject"), &pUnknown);

    IMyOptionObjectPtr pMyOptionObject(pUnknown);
    CCfString strProperty;
    pMyOptionObject->GetMyProperty(&strProperty);
}
```

See also

IRuntime (Page 7988)

IErrorInfo (Page 8001)

IRuntime

Description

The C++ interface "IRuntime" specifies methods for information and the addressing of the Runtime system.

Members

The following methods are specified in the interface:

"GetObject" method

Create new instance of an object of the Runtime system. Possible object types are defined in the configuration file OdkObjectModel.xml.

```
CFRESULT GetObject(const CFSTR value, ICfUnknown **ppObject)
```

- `value`
  [in]: Name of the object type, for example "Tag" for tags
- `ppObject`
  [out]: Points to the initialized object of the runtime system.

"GetProduct" method

Return an "IProduct" object that allows access to the version information and installed options of the Runtime system.

```
CFRESULT GetProduct(IProduct **ppProduct)
```

`ppProduct`
[in/out]: Points to an "IProduct" object that contains the product information of the runtime system.

"GetOption" method

Referencing installed option of the Runtime system.

```
CFRESULT GetOption(
         const CFSTR optionName,
         IOption **ppOption)
```

- `optionName`
  [in]: Name of the installed option
- `ppProduct`
  [out]: Points to an installed option of the Runtime system as "IOption" object.

"GetUserName" method

Return the name of the logged-on user.

```
CFRESULT GetUserName(CFSTR* name)
```

`name`
[out]: Displays the user name.

Example

Initialize an object of the "Tag" type of the Runtime system:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    ITagPtr pTag(pUnk);
    pTag->SetTagName(tag);
    ... //further tag processing
}
```

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
   IProductPtr pProduct;
   pRuntime->GetProduct(&pProduct);

   uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
   IVersionInfoPtr pVersion;

   pProduct->GetVersion(&pVersion);
   pVersion->GetMajor(&uintMajor);
   pVersion->GetMinor(&uintMinor);
   pVersion->GetServicePack(&uintServicePack);
   pVersion->GetUpdate(&uintUpdate);
   wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
uintServicePack << L"-" << uintUpdate << endl;
}
```

Use installed options:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnk;
    pOdkOption->GetObject("MyOptionObject2", &pUnk);

    IMyOptionObjectPtr pMyOptionObject(pUnknown);
    CCfString strProperty;
    pMyOptionObject->GetMyProperty(&strProperty);

    //using extension methods for CPM node
    ICpmPtr pCpm;
    pRuntime->GetObject(CCfString("Cpm"), (ICfUnknown**) &pCpm);

    ICpmNodePtr pCpmNode;
    CCfString strNode(".hierarchy::PlantView\\Unit1");
    pCpm->GetNode(strNode, &pCpmNode);

    //using specific option interface
    IMyOptionPtr pMyOption(pOdkOption);

    IMyCpmNodeFormulaPtr pFormula;
    pMyOption->GetObject(pCpmNode, CCfString("Formula"), (ICfUnknown**) &pFormula);
    pFormula->SetName(CCfString("Quality"));
    int32_t result;
    pFormula->Calc(&result);
}
```

See also

IOdkRt (Page 7986)

IProduct (Page 7991)

IOption (Page 7993)

IProduct

Description

The C++ interface "IProduct" specifies methods for handling product information of the Runtime system.

Members

The following methods are specified in the interface:

"GetOptions" method

Return installed options of the Runtime system as array of "IOption" objects.

```
CFRESULT GetOptions(IOptionEnumerator **ppEnumerator)
```

`ppEnumerator`
[out]: Points to the installed options as "IOptionEnumerator" object.

"GetVersion" method

Return version structure of the installed Runtime system as "IVersionInfo" object.

```
CFRESULT GetVersion(IVersionInfo** versionInfo)
```

`versionInfo`
[out]: Points to a structure with version information of the installed Runtime system.

Example

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
   IProductPtr pProduct;
   pRuntime->GetProduct(&pProduct);
   uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
   IVersionInfoPtr pVersion;

   pProduct->GetVersion(&pVersion);
   pVersion->GetMajor(&uintMajor);
   pVersion->GetMinor(&uintMinor);
   pVersion->GetServicePack(&uintServicePack);
   pVersion->GetUpdate(&uintUpdate);
   wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
uintServicePack << L"-" << uintUpdate << endl;
}
```

Output name of all installed options:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
   IProductPtr pProduct;
   pRuntime->GetProduct(&pProduct);
   IOptionEnumeratorPtr pItems;

   while (pItems->MoveNext() == CF_SUCCESS)
   {
      IOptionPtr pValue;
      pItems->Current(&pValue);
      CCfString module;
      pValue->GetName(&module);
      wcout << L"Option name: " << module << endl;
   }
}
```

See also

IOdkRt (Page 7986)

IRuntime (Page 7988)

IOption (Page 7993)

IOptionEnumerator (Page 7994)

IVersionInfo (Page 7996)

IOption

Description

The C++ interface "IOption" specifies properties and methods for handling installed product options of the Runtime system.

Members

The following methods are specified in the interface:

"GetName" method

Return name of an installed option of the Runtime system.

```
CFRESULT GetName(CFSTR *pValue)
```

`pValue`
[out]: Points to the name of an installed option of the runtime system.

"GetObject" method

Referencing installed option of the Runtime system.

```
CFRESULT GetObject(
         const CFSTR Value,
         ICfUnknown** ppObject)
```

- `Value`
  [in]: Name of the installed option of the Runtime system
- `ppObject`
  [out]: Points to the installed option of the Runtime system as an "ICfUnknown" object.

"GetVersion" method

Reference version structure of an installed option of the Runtime system as "IVersionInfo" object.

```
CFRESULT GetVersion(IVersionInfo** versionInfo)
```

`versionInfo`
[out]: Points to a structure with version information of an installed option of the Runtime system.

Example

Read out installed option:

Copy code

```
void GetOptionObject(IRuntimePtr pRuntime)
{
    Siemens::Runtime::HmiUnified::Common::Cpp::IOptionPtr pOdkOption;
    pRuntime->GetOption(CCfString("MyOptionName"), &pOdkOption);

    ICfUnknownPtr pUnk;
    pOdkOption->GetObject("MyOptionObject2", &pUnk);

    IMyOptionObjectPtr pMyOptionObject(pUnknown);
    CCfString strProperty;
    pMyOptionObject->GetMyProperty(&strProperty);
}
```

See also

IProduct (Page 7991)

IOptionEnumerator (Page 7994)

IVersionInfo (Page 7996)

IOptionEnumerator

Description

The "IOptionEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of installed product options of the Runtime system.

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IOption **ppItem)
```

ppItem
[out]: Points to the current "IOption" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext"method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

value
[out]: Points to the value for the number of the elements of the list.

Example

Access the installed options "IOption" of the runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
   IProductPtr pProduct;
   pRuntime->GetProduct(&pProduct);

   pProduct->GetOptions(&pItems);
   IOptionEnumeratorPtr pItems;

   while (pItems->MoveNext() == CF_SUCCESS)
   {
      IOptionPtr pValue;
      pItems->Current(&pValue);
      ...
   }
}
```

See also

IOption (Page 7993)

IVersionInfo

Description

The C++ interface "IVersionInfo" specifies methods for reading out version information of the runtime system.

Members

The following methods are specified in the interface:

"GetMajor" method

Return main version of an installed option of the Runtime system.

```
CFRESULT GetMajor(uint16_t *pValue)
```

`pValue`
[out]: Points to the main version of an installed option of the Runtime system.

"GetMinor" method

Return minor version of an installed option of the Runtime system.

```
CFRESULT GetMinor(uint16_t *pValue)
```

`pValue`
[out]: Points to the minor version of an installed option of the Runtime system.

"GetServicePack" method

Return service pack of an installed option of the Runtime system.

```
CFRESULT GetServicePack(uint16_t *pValue)
```

`pValue`
[out]: Points to the service pack of an installed option of the Runtime system.

"GetUpdate" method

Return update version of an installed option of the Runtime system.

```
CFRESULT GetUpdate(uint16_t *pValue)
```

`pValue`
[out]: Points to the update version of an installed option of the Runtime system.

Example

Output technical product version of the Runtime system:

Copy code

```
void GetVersionInfo(IRuntimePtr pRuntime)
{
   IProductPtr pProduct;
   pRuntime->GetProduct(&pProduct);

   uint16_t uintMajor, uintMinor, uintUpdate, uintServicePack;
   IVersionInfoPtr pVersion;

   pProduct->GetVersion(&pVersion);
   pVersion->GetMajor(&uintMajor);
   pVersion->GetMinor(&uintMinor);
   pVersion->GetServicePack(&uintServicePack);
   pVersion->GetUpdate(&uintUpdate);
   wcout << L"WinCC Unified version: " << uintMajor << L"-" << uintMinor << L"-" <<
uintServicePack << L"-" << uintUpdate << endl;
}
```

See also

IProduct (Page 7991)

IOption (Page 7993)

IErrorResult

Description

The "IErrorResult" interface is a C++ interface that specifies methods for reading out error details.

Members

The following methods are specified in the interface:

"GetError" method

Read out error code of an error message.

```
CFRESULT GetError(CFRESULT *value)
```

`value`
[out]: Points to an error code.

"GetName" method

Read out name of the associated object of the data source.

```
CFRESULT GetError(CFSTR *value)
```

`value`
[out]: Points to an object name.

Example

Read out details of "IErrorResult" error messages:

Copy code

```
IErrorResultEnumerator* WriteTagSetSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
    IErrorResultEnumerator* pEnumerator = nullptr;

    ITagSetPtr pTagSet(pUnk);

    // add tags to tag set
    for (int i = 0; i < tags.size(); i++)
    {
       pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
    }

    errCode = pTagSet->Write(&pEnumerator);
    if (CF_FAILED(errCode))
    {
       std::wcout << L"Write operation failed." << std::endl;
       PrintErrorInformation(errCode, L"Write", pRuntime);
    }
    if (pEnumerator != nullptr)
    {
       while (pEnumerator->MoveNext() == CF_SUCCESS)
       {
          IErrorResult* pValue;
          CFRESULT errorCode = pEnumerator->Current(&pValue);
          if (pValue != nullptr && CF_SUCCEEDED(errorCode))
          {
             pValue->GetError(&errorCode);
             CCfString str;
             pValue->GetName(&str);
             if (CF_FAILED(errorCode))
             {
                std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode:" <<
errorCode << std::endl;
                PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
             }
          }
       }
    }
    return pEnumerator;
}
```

See also

IErrorResultEnumerator (Page 7999)

IErrorResultEnumerator

Description

The "IErrorResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of error messages of the Runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IErrorResult **ppItem)
```

`ppItem`
[out]: Points to the current "IErrorResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

Example

Access the "IErrorResult" error messages when writing a TagSet:

Copy code

```
IErrorResultEnumerator* WriteTagSetSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
    ICfUnknownPtr pUnk;
    CFRESULT errCode = pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
    IErrorResultEnumerator* pEnumerator = nullptr;

    ITagSetPtr pTagSet(pUnk);

    // add tags to tag set
    for (int i = 0; i < tags.size(); i++)
    {
       pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
    }

    errCode = pTagSet->Write(&pEnumerator);
    if (CF_FAILED(errCode))
    {
       std::wcout << L"Write operation failed." << std::endl;
       PrintErrorInformation(errCode, L"Write", pRuntime);
    }
    if (pEnumerator != nullptr)
    {
       while (pEnumerator->MoveNext() == CF_SUCCESS)
       {
          IErrorResult* pValue;
          CFRESULT errorCode = pEnumerator->Current(&pValue);
          if (pValue != nullptr && CF_SUCCEEDED(errorCode))
          {
             pValue->GetError(&errorCode);
             CCfString str;
             pValue->GetName(&str);
             if (CF_FAILED(errorCode))
             {
                std::wcout << L"Write Tag failed, Tag name: " << str << L", ErrorCode:" <<
errorCode << std::endl;
                PrintErrorInformation(errorCode, L"Write Tag", pRuntime);
             }
          }
       }
    }
    return pEnumerator;
}
```

See also

IErrorResult (Page 7997)

ITagSet (Page 8014)

ITagSetQCD (Page 8019)

IErrorInfo

Description

The "IErrorInfo" interface is a C++ interface that specifies methods for handling error codes.

Members

The following methods are specified in the interface:

"GetErrorDescription" method

Output an error description for the error code.

You have to use the "GetErrorHandler" method to instantiate an "IErrorInfo" object beforehand.

```
CFRESULT GetErrorDescription(
         uint32_t errorCode,
         CFSTR *errorDescription)
```

- `errorCode`
  [in]: Error code that is handed over by the ODK client.
- `errorDescription`
  [out]: Points to the error description of the error code.

Example

Output object name and description of an error:

Copy code

```
void PrintErrorInformation(CFRESULT errorCode, CCfSmartString objectName, IRuntimePtr
pRuntime)
{
   if (pRuntime != nullptr)
   {
      IErrorInfoPtr pInfo;
      CFRESULT result = pRuntime->GetObject(CCfString(L"ErrorHandler"),
(ICfUnknown**)&pInfo);
      if (CF_FAILED(result))
      {
         std::wcout << "Error occurred: Can not create 'ErrorHandler' object " << std::endl;
         return;
      }
      CCfString resStr;
      result = pInfo->GetErrorDescription(errorCode, &resStr);
      if (CF_SUCCEEDED(result))
      {
         CCfSmartString errorDescription(resStr);
         std::wcout << "Error occurred: '" << errorDescription.Get() << "', ObjectName = "
<< objectName.Get() << std::endl;
      }
      else
      {
         std::wcout << "Error occurred: 'GetErrorDescription' failed, Error number: " <<
result << std::endl;
      }
   }
   else
   {
      CCfString strMsg(L"");
      Siemens::Runtime::HmiUnified::Common::IErrorInfo* pErrorInfo;
      GetErrorHandler(&pErrorInfo);
      CFRESULT rVal = pErrorInfo->GetErrorDescription(errorCode, &strMsg);
      CCfSmartString errorDescription(strMsg);
      std::wcout << "Error occurred: '" << errorDescription.Get() << "', ObjectName = " <<
objectName.Get() << std::endl;
   }
}
```

See also

IOdkRt (Page 7986)

20.3.9.3 Interfaces of the tags

IProcessValue

Description

The C++ interface "IProcessValue" specifies properties and methods for values of process tags of the Runtime system. The "IProcessValue" interface represents values from the result of a read operation or monitoring.

Members

The following methods are specified in the interface:

"GetTagName" method

Return the name of the tag.

```
CFRESULT GetTagName(CFSTR *value)
```

`value`
[out]: Points to the name of the tag belonging to the process value.

"GetValue" method

Return value of the tag at the moment of the read operation.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the tag.

"GetQuality" method

Return quality code of the read operation of the tag.

```
CFRESULT GetQuality(int32_t *value)
```

`value`
[out]: Points to the quality code of the process tag.

"GetTimeStamp" method

Return the time stamp of the last successful read operation of the tag.

```
CFRESULT GetTimeStamp(CFDATETIME64 *value)
```

`value`
[out]: Points to the time stamp of the read operation of the process tag.

"GetError" method

Return error code of the last read or write operation of the tag.

```
CFRESULT GetError(int32_t *value)
```

`value`
[out]: Points to the error code of the process tag.

Example

Read out a process tag and output the properties of the "IProcessValue" object:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

   ITagPtr pTag(pUnk);
   pTag->SetTagName(tag);
   IProcessValuePtr pValue;

   // Read value of tag
   pTag->Read(&pValue);

   CCfString timeStamp;
   CFDATETIME64 cfTimeStamp;
   pValue->GetTimeStamp(&cfTimeStamp);
   CCfDateTime64 time(cfTimeStamp);
   timeStamp = time.GetDateTimeString(false);
   CCfString strName;
   pValue->GetTagName(&strName);
   CCfVariant varValue;
   pValue->GetValue(&varValue);
   int32_t quality;
   pValue->GetQuality(&quality);

   std::wcout << strName.ToUTF8().c_str() << L" " << timeStamp.ToUTF8().c_str() << L" " <<
L" Value: " << (double)(varValue) << L" Quality: " << quality << std::endl;
}
```

See also

IProcessValueEnumerator (Page 8004)

ITag (Page 8006)

ITagSet (Page 8014)

ITagSetQCD (Page 8019)

IProcessValueEnumerator

Description

The "IProcessValueEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of process values of the Runtime system. The enumeration is, for example, used when reading out process values of a TagSet.

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IProcessValue **ppItem)
```

`ppItem`
[out]: Points to the current "IProcessValue" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext"method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of the elements of the list.

Example

Output process values of TagSets:

Copy code

```
...
IProcessValueEnumeratorPtr pItems;
pTagSet->Read(&pItems);

std::wcout << "Read finished " << std::endl;
// Iterate over the process value objects
while(CF_SUCCEEDED(pItems->MoveNext()))
{
   IProcessValuePtr pValue;
   pItems->Current(&pValue);   // get current process value

   CCfString timeStamp;
   CFDATETIME64 cfTimeStamp;
   pValue->GetTimeStamp(&cfTimeStamp);
   CCfDateTime64 time(cfTimeStamp);
   timeStamp = time.GetDateTimeString(false);
   CCfString strName;

   pValue->GetTagName(&strName);
   CCfVariant varValue;
   pValue->GetValue(&varValue);
   int32_t quality;
   pValue->GetQuality(&quality);
   int32_t error = 0;
   pValue->GetError(&error);
   std::wcout << L" " << strName.ToUTF8().c_str() << L" " << timeStamp.ToUTF8().c_str() <<
L" " << L" Value: " << static_cast<double>(varValue) << L" Quality: " << quality << L"
Error: " << (uint32_t)error << std::endl;
}
...
```

See also

IProcessValue (Page 8003)

ITag

Description

The C++ interface "ITag" specifies methods for handling tags of the Runtime system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetTagName" method

Set name of the tag.

```
CFRESULT SetTagName(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag

"Write" method

Write process value of the tag synchronously in the Runtime system.

```
CFRESULT Write(
         const CFVARIANT value,
         CFENUM type = HmiWriteType::NoWait)
```

- `value`
  [in]: Tag value
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType::NoWait` (default): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType::Wait`: Waits until the tag value is written in the AS. The associated errors are written.

"WriteQCD" method

Write process value with quality code of the tag synchronously in the Runtime system. The tag also has a freely definable time stamp. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

```
CFRESULT WriteQCD(
         const CFVARIANT value,
         const CFDATETIME64 timeStamp,
         const int16_t qualityCode,
         CFENUM type = HmiWriteType::NoWait)
```

- `value`
  [in]: Tag value
- `timeStamp`
  [in]: Time stamp of the tag. Also in the past.

- `qualityCode`
  [in]: Quality code of the tag
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType::NoWait` (default): Writes the tag value without waiting. Errors for the write operation are not detected.
  - `HmiWriteType::Wait`: Waits until the tag value is written in the AS. The associated errors are written.

"WriteWithOperatorMessage" method

Write process value of the tag synchronously in the runtime system and create operator input alarm. In addition to the reason, the operator input alarm contains the old and new value, the user and host names and the unit.

```
CFRESULT WriteWithOperatorMessage(
         const CFVARIANT value,
         const CFSTR reason)
```

- `value`
  [in]: Value of the tag
- `reason`
  [in]: Reason for the value change for alarm

"Read" method

Read process value and properties of the tag synchronously from the Runtime system.

```
CFRESULT Read(
         IProcessValue **ppValue,
         CFENUM type = HmiReadType::Cache)
```

- `ppValue`
  [out]: Points to the properties and the value of the tag as an "IProcessValue" object.
- `type`
  [in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
  - `HmiReadType::Cache` (default): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
  - `HmiReadType::Device`: Reads the tag value directly from the AS. The tag image is not used.

Example

Write tags synchronously:

Copy code

```
CFRESULT WriteSingleTagSync(IRuntimePtr pRuntime, CCfString tag, CCfVariant& value)
{
   ICfUnknownPtr pUnk;
   CFRESULT errCode = pRuntime->GetObject(CCfString(L"Tag"), &pUnk);
   ITagPtr pTag(pUnk);
   pTag->SetTagName(tag);
   // Write value of tag
   errCode = pTag->Write(value, HmiWriteType::Wait);
   return errCode;
}
```

Write tag with time stamp and quality code synchronously:

Copy code

```
void WriteSingleTagQCDSync(IRuntimePtr pRuntime, CCfString tag, CCfVariant& value)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"Tag"), &pUnk);
   ITagPtr pTag(pUnk);
   pTag->SetTagName(tag);
   // Write value of tag
   pTag->WriteQCD(value, CCfDateTime64::Now(), 128, HmiWriteType::Wait);
}
```

Read tags synchronously:

Copy code

```
CFRESULT ReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
    ICfUnknownPtr pUnk;
    pRuntime->GetObject(CCfString(L"Tag"), &pUnk);

    ITagPtr pTag(pUnk);
    pTag->SetTagName(tag);

    IProcessValuePtr pValue;

    // Read value of tag
    pTag->Read(&pValue);
    ...

    return errCode;
}
```

See also

IProcessValue (Page 8003)

ITagCallback (Page 8010)

ITagCallback

Description

The "ITagCallback" interface and the "COdkTagSourceCBBase" and "COdkTagSetCB" classes define methods for implementing asynchronous read and write operations with tags. The methods are used by the C++-interface "ITagSet".

Members of the interface

The following methods are specified in the "ITagCallback" interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

The "OnReadComplete" callback method is called when the "ITagSet.ReadAsync" method is used.

```
CFRESULT OnReadComplete(
         IProcessValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId,
         CFBOOL completed)
```

- `pEnumerator`
  [out]: Points to an "IProcessValueEnumerator" object that contains the enumeration of the read process values.
- `systemError`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the tag.
- `completed`
  [out]: Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are read out yet.

"OnWriteComplete" method

Callback method is called on completion of asynchronous write operations.

The "OnWriteComplete" callback method is called when the "ITagSet.WriteAsync" method is used.

```
CFRESULT OnWriteComplete(
         IErrorResultEnumerator *pEnumerator,
         uint32_t errorCode,
```

```
        int32_t contextId
        CFBOOL completed)
```

- `pEnumerator`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations of the tag.
- `systemError`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the tag.
- `completed`
  [out]: Status of the asynchronous transfer:
  - True: All alarms are read out.
  - False: Not all alarms are read out yet.

"OnDataChanged" method

Callback method is called when a monitored tag value is changed.

The callback method is called after the process value change of a monitored TagSet ("ITagSet.Subscribe" method).

```
CFRESULT OnDataChanged(
        IProcessValueEnumerator *pEnumerator,
        uint32_t errorCode,
        int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "IProcessValueEnumerator" object that contains the enumeration of the read process values.
- `systemError`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the tag.

Members of the classes

The following methods are implemented in the "COdkTagSourceCBBase" and "COdkTagSetCB" classes:

"SetEvent" method

Signals an event.

```
CFBOOL SetEvent()
```

"ResetEvent" method

Resets the signaling of an event.

```
CFBOOL ResetEvent()
```

"WaitForcompletion" method

Waits for the signaling of an event.

```
uint32_t WaitForcompletion(uint32_t dwMilliseconds)
```

`dwMilliseconds`
[in]: Time interval in milliseconds for which an event is waited for.

"GetValues" method

Return process values of the asynchronous read operation.

```
std::vector<IProcessValue*> GetValues()
```

Example

In the following section tags in a TagSet are read asynchronously. To this purpose the "ReadTagSetAsync" function uses a "COdkTagSetCB" object that implements the "ITagCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "COdkTagSetCB" object is determined via reference counting.

Copy code

```
void ReadTagSetAsync(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
   ITagSetPtr pTagSet(pUnk);
   // add tags to tag set
   for(int i = 0; i < tags.size(); i++)
   {
      pTagSet->Add(tags[i]);
   }

   COdkTagSetCB* pTagSetCB = new COdkTagSetCB();
   pTagSetCB->AddRef();
   // Read the tag set asynchronously, result comes via callback
   pTagSet->ReadAsync(pTagSetCB);
   pTagSetCB->WaitForcompletion(std::numeric_limits<uint32_t>::max();
   vector<IProcessValuePtr> pValues = pTagSetCB->GetValues();
   std::wcout << L"Read finished " << std::endl;

    // display tag values
   for(int i = 0; i < pValues.size(); i++)
   {
      IProcessValue* pValue = pValues[i];
      CCfString timeStamp;
      CFDATETIME64 cfTimeStamp;
      pValue->GetTimeStamp(&cfTimeStamp);
      CCfDateTime64 time(cfTimeStamp);
      timeStamp = time.GetDateTimeString(false);
      CCfString strName;
      pValue->GetTagName(&strName);
      CCfVariant varValue;
      pValue->GetValue(&varValue);
      int32_t quality;
      pValue->GetQuality(&quality);
      int32_t error = 0;
      pValue->GetError(&error);
      std::wcout << L" " << strName.ToUTF8().c_str() << L" " << timeStamp.ToUTF8().c_str()
<< L" " << L" Value: " << static_cast<double>(varValue) << L" Quality: " << quality << L"
Error: " << (uint32_t)error << std::endl;
   }
}
```

See also

ITag (Page 8006)

ITagSet (Page 8014)

ITagSetQCD (Page 8019)

ITagSet

Description

The C++ interface "ITagSet" specifies properties and methods for an optimized access to several tags of the Runtime system.

After initialization of the "ITagSet" object, you can execute read and write access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

`value`
[in]: ContextId of the tag:

"GetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

`value`
[out]: Points to the ContextId of the tag.

"Remove" method

Remove individual tag from a TagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag that is removed from TagSet.

"Add" method

Add tag to a TagSet.

```
CFRESULT Add(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag for TagSet

"AddWithValue" method

Add tag with process value to the TagSet.

```
CFRESULT AddWithValue(const CFSTR tagName, const CFVARIANT value)
```

- `tagName`
  [in]: Name of the tag
- `value`
  [in]: New value of the tag

"GetValue" method

Read process value of a tag of a TagSet.
To fill the local TagSet with process values, a "Read", "ReadAsync" or "AddWithValue" method must be called beforehand.

The values of the "IProcessValue" object are not available until after execution of the methods "Read", "ReadAsync" or "AddWithValue".

```
CFRESULT GetValue(const CFSTR tagName, CFVARIANT *pValue)
```

- `tagName`
  [in]: Name of the tag from the TagSet
- `pValue`
  [out]: Points to the process value of the tag.

"SetValue" method

Change the process value of a tag of a TagSet.

The "SetValue" method changes only the values of the local TagSet. In order to write the changed values into the automation system you must additionally execute the "Write" or "WriteAsync" method.

```
CFRESULT SetValue(const CFSTR tagName, const CFVARIANT value)
```

- `tagName`
  [in]: Name of the tag from the TagSet
- `value`
  [in]: New value of the tag

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator
**ppEnumerator, CFENUM type = HmiWriteType::NoWait)
```

- `ppEnumerator`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
  - `Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteWithOperatorMessage" method

Write process values of all tags of a TagSet synchronously in the Runtime system and create operator input alarms. In addition to the reason, the operator input alarms contain the old and new value, the user and host names and the unit.

```
CFRESULT WriteWithOperatorMessage(const CFSTR reason)
```

`reason`
[in]: Reason for the value change for alarm

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType::Wait` type and waits until the tag value has been written in the automation system. Associated errors are written.

```
CFRESULT WriteAsync(
         ITagCallback* pTagSetCb)
```

- `pTagSetCb`
  [in]: Points to the "ITagCallback" object that implements the callback interface.

"Read" method

Read process values and properties of all the tags of a TagSet synchronously from the Runtime system.

```
CFRESULT Read(
         IProcessValueEnumerator **ppEnumerator,
         CFENUM type = HmiReadType::Cache)
```

- `ppEnumerator`
  [in/out]: Points to the properties and process values of the tags as an "IProcessValueEnumerator" object.
- `type`
  [in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
  - `Cache` (default): Reads the tag values from the tag image. If no subscription exists, the tag is subscribed.
  - `Device`: Reads the tag values directly from the automation system. The tag image is not used.

"ReadAsync" method

Read process values and properties of all the tags of a TagSet asynchronously from the Runtime system.

```
CFRESULT ReadAsync(
         ITagCallback *pTagSetCb,
         CFENUM type = HmiReadType::Cache)
```

- `pTagSetCb`
  [in]: Points to the "ITagCallback" object that implements the callback interface.
- `type`
  [in/optional]: The enumeration "HmiReadType" specifies the origin of the tag value:
  - `Cache` (default): Reads the tag value from the tag image. If no subscription exists, the tag is subscribed.
  - `Device`: Reads the tag value directly from the AS. The tag image is not used.

"Subscribe" method

Subscribe all tags of a TagSet asynchronously for cyclic monitoring of the process values.

Note

Tags from IO devices with the "Cyclic in operation" acquisition mode

For a tag with the acquisition mode "Cyclic in operation", the value stored in the process image when Subscribe is called might be outdated. OnAdd therefore only provides the QualityCode "uncertain". Only value changes made after the Subscribe call provide the current value and the QualityCode "good".

```
CFRESULT Subscribe(ITagCallback *pTagSetCb)
```

`pTagCb`
[in]: Points to the "ITagCallback" object that implements the callback interface.

"CancelSubscribe" method

Cancel monitoring of all tags of a TagSet.

```
CFRESULT CancelSubscribe()
```

"GetCount" method

Return the number of tags of a TagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`
[out]: Points to the value for the number of tags of the TagSet list.

"Clear" method

Remove all tags from a TagSet.

```
CFRESULT Clear()
```

Example

Write TagSet asynchronously:

Copy code

```
struct TagTuple_T
{
    CCfSmartString _tagName;
    CCfVariant _tagValue;
};

void WriteTagSetAsync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
   vector<IErrorResultPtr> pErrors;
   ITagSetPtr pTagSet(pUnk);

   // add tags to tag set
   for(int i = 0; i < tags.size(); i++)
   {
      pTagSet->AddWithValue(CCfString(tags[i]._tagName), tags[i]._tagValue);
   }

   COdkTagSetCB* pTagSetCB = new COdkTagSetCB();
   pTagSetCB->AddRef();
   // Write value of tag asynchronously
   pTagSet->WriteAsync(pTagSetCB);
   pTagSetCB->WaitForcompletion(std::numeric_limits<uint32_t>::max());
}
```

Start monitoring for tags of a TagSet:

Copy code

```
void SubscribeTagSet(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
   ICfUnknown* pUnk;
   pRuntime->GetObject(CCfString(L"TagSet"), &pUnk);
   ITagSetPtr pTagSet(pUnk);
   // add tags to tag set
   for(int i = 0; i < tags.size(); i++)
   {
      pTagSet->Add(tags[i]);
   }
   COdkTagSetCB* pTagSetCB = new COdkTagSetCB();
   pTagSetCB->AddRef();
   // subscribe tags
   pTagSet->Subscribe(pTagSetCB);
   pTagSetCB->WaitForcompletion(1500);
   std::wcout << L"Stop subscribtion." << std::endl;
   pTagSet->CancelSubscribe();
}
```

See also

IProcessValue (Page 8003)

ITagCallback (Page 8010)

IErrorResultEnumerator (Page 7999)

ITagSetQCD

Description

The C++ interface "ITagSetQCD" specifies methods for optimized writing of several tags of the Runtime system. The tags also have a freely definable time stamp and quality code. You can use this to acquire past external measured values, for example.

Note

Reaction to external tags

For external tags, the method only writes the tag value. The QualityCode and time stamp are set internally by the system.

After initialization of the "ITagSetQCD" object, you can have read access to multiple tags in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

`value`
[in]: ContextId of the tag:

"GetContextId" method

Set ID as an additional identification feature of the tag. The ContextId can, for example, be used to recognize identically named tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t *value)
```

`value`
[out]: Points to the ContextId of the tag.

"Remove" method

Remove individual tag from a TagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

`tagName`
[in]: Name of the tag that is removed from TagSet.

"Add" method

Add tag with process value, quality code and time stamp to the TagSet.

```
CFRESULT Add(
         const CFSTR tagName,
         const CFVARIANT value,
         const CFDATETIME64 timeStamp,
         const int16_t qualityCode)
```

- `tagName`
  [in]: Name of the tag for TagSet
- `value`
  [in]: New process value of the tag
- `timeStamp`
  [in]: Time stamp of the process value. Also in the past.
- `qualityCode`
  [in]: Quality code for process value

"Write" method

Write process values of all tags of a TagSet synchronously in the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator
**ppEnumerator, CFENUM type = HmiWriteType::NoWait)
```

- `ppEnumerator`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.
- `type`
  [in/optional]: The enumeration "HmiWriteType" specifies whether the method waits for the write operation to be completed:
  - `HmiWriteType::NoWait` (default): Writes the tag values without waiting. Errors for the write operation are not detected.
  - `HmiWriteType::Wait`: Waits until the tag values are written in the automation system. The associated errors are written.

"WriteAsync" method

Write process values of all tags of a TagSet asynchronously in the Runtime system.

The method always has the `HmiWriteType::Wait` type and waits until the tag value has been written in the automation system. Associated errors are written.

```
CFRESULT WriteAsync(
         ITagCallback* pTagSetCb)
```

- `pTagSetCb`
  [in]: Points to the "ITagCallback" object that implements the callback interface.

"GetCount" method

Return the number of tags of a TagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`
[out]: Points to the value for the number of tags of the TagSet list.

"GetItem" method

Return tag of the TagSet for changing or reading out process value, QualityCode and time stamp.

```
CFRESULT GetItem(
         const CFSTR name,
         ITagSetQCDItem **pTagSetQCDItem)
```

- `name`
  [in]: Name of the tag in the TagSet
- `pTagSetQCDItem`
  [out]: Points to tag as "TagSetQCDItem" object

"Clear" method

Remove all tags from a TagSet.

```
CFRESULT Clear()
```

Example

Write TagSet with time stamp and quality code synchronously:

Copy code

```
struct TagTuple_T
{
    CCfSmartString _tagName;
    CCfVariant _tagValue;
};


void WriteTagSetQCDSync(IRuntimePtr pRuntime, std::vector<TagTuple_T> tags)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"TagSetQCD"), &pUnk);
   ITagSetQCDPtr pTagSetQCD(pUnk);

   // add tags to tag set
   for(int i = 0; i < tags.size(); i++)
   {
     pTagSetQCD->Add(CCfString(tags[i]._tagName), tags[i]._tagValue, CCfDateTime64::Now(),
128);
   }
   IErrorResultEnumerator* pEnumerator;
   pTagSetQCD->Write(&pEnumerator);
   while (pEnumerator->MoveNext() == CF_SUCCESS)
   {
      IErrorResult* pValue;
      CFRESULT errorCode = pEnumerator->Current(&pValue);
      pValue->GetError(&errorCode);
      CCfString str;
      pValue->GetName(&str);
   }
}
```

See also

IProcessValue (Page 8003)

ITagCallback (Page 8010)

ITagSetQCDItem (Page 8023)

IErrorResultEnumerator (Page 7999)

ITagSetQCDItem

Description

The C++ interface "ITagSetQCDItem" specifies methods for adapting tags of the Runtime system. You can read in tags into a TagSetQCD and then change all names, values, time stamps and QualityCodes of the tags.

Note

Reaction to external tags

For external tags, only the tag value is set. The QualityCode and time stamp are set internally by the system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"GetName" method

Return the name of the tag.

```
CFRESULT GetName(CFSTR *name)
```

`name`
[out]: Points to the name of the tag.

"SetName" method

Change name of the tag.

```
CFRESULT SetName(const CFSTR name)
```

`name`
[in]: New name of the tag.

"GetValue" method

Return value of the tag at the moment of the read operation.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the tag.

"SetValue" method

Change value of the tag.

```
CFRESULT SetValue(const CFVARIANT value)
```

`value`
[in]: New process value of the tag.

"GetTimeStamp" method

Return time stamp of the tag.

```
CFRESULT GetTimeStamp(CFDATETIME64 *timeStamp)
```

`timeStamp`
[out]: Points to the time stamp of the tag.

"SetTimeStamp" method

Change time stamp of the tag.

```
CFRESULT SetTimeStamp(const CFDATETIME64 timeStamp)
```

`timeStamp`
[in]: New time stamp of the tag

"GetQuality" method

Return quality code of the read operation of the tag.

```
CFRESULT GetQuality(int32_t *qualityCode)
```

`qualityCode`
[out]: Points to the quality code of the tag.

"SetQuality" method

Change quality code of the tag.

```
CFRESULT SetQuality(const int32_t qualityCode)
```

`qualityCode`
[in]: New quality code of the tag

See also

ITagSetQCD (Page 8019)

ILoggedTagValue

Description

The C++ interface "ILoggedTagValue" specifies the properties that a logged process value of a logging tag has in the logging system.

An "ILoggedTagValue" instance is a pure data object. The instance encapsulates all properties of the logged process value. It represents a historical process value.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"GetTagName" method

Return name of the logging tag.

```
CFRESULT GetTagName(CFSTR *value)
```

`value`
[out]: Points to the name of the process value belonging to the logging tag.

"GetValue" method

Return process value of the logging tag.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the logging tag.

"GetQuality" method

Return quality code of the process value.

```
CFRESULT GetQuality(int16_t *value)
```

`value`
[out]: Points to the quality code of the process value.

If the "ILoggedTagValue" instance has been added to the log by calling the "Write" method, "Quality" has the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

"GetTimeStamp" method

Return time stamp of the process value.

```
CFRESULT GetTimeStamp(CFDATETIME64 *value)
```

`value`
[out]: Points to the time stamp of the process value.

"GetError" method

Return error code of the process value.

```
CFRESULT GetError(uint32_t *value)
```

`value`
[out]: Points to the error code of the process value.

"GetFlags" method

Return context information from the read operation for the process value.

```
CFRESULT GetFlags(HmiTagLoggingValueFlags *value)
```

`value`
[out]: Points to the context information.

The "HmiTagLoggingValueFlags" enumeration contains the following bit-by-bit-coded values:

- 0: `Extra`
  There are still additional values at the time of the process value.
- 2: `Calculated`
  Process value is calculated.
- 16: `Bounding`
  Process value is a limit value.
- 32: `NoData`
  No additional information available
- 64: `FirstStored`
  Process value is the first value stored in the logging system.
- 128: `LastStored`
  Process value is the last value stored in the logging system.

Example

Output process values of a logging tag:

Copy code

```
void PrintValues(ILoggedTagValueEnumeratorPtr pItems,IRuntimePtr pRuntime)
{
   // Iterate over the process value objects
   while (CF_SUCCEEDED(pItems->MoveNext()))
   {
      ILoggedTagValuePtr pValue;
      pItems->Current(&pValue);   // get current process value
      uint32_t nerror;
      pValue->GetError(&nerror);
      if (nerror)
      {
         PrintErrorInformation(nerror, L"GetObject", pRuntime);
         CCfString strName;
         pValue->GetTagName(&strName);
         std::wcout << "Tag With Error: "<<strName.ToUTF8().c_str() << std::endl;
         continue;
      }
      CCfString timeStamp;
      CFDATETIME64 cfTimeStamp;
      pValue->GetTimeStamp(&cfTimeStamp);
      CCfDateTime64 time(cfTimeStamp);
      timeStamp = time.GetDateTimeString(false);
      CCfString strName;
      pValue->GetTagName(&strName);
      CCfVariant varValue;
      pValue->GetValue(&varValue);
      HmiTagLoggingValueFlags enumflag;
      pValue->GetFlags(&enumflag);
      std::wcout << strName.ToUTF8().c_str() << L" " << timeStamp.ToUTF8().c_str() << L"
Value: " << (double)(varValue) <<" Flagvalue = "<< enumflag<< std::endl;
   }
}
```

See also

ILoggedTagValueEnumerator (Page 8027)

ILoggedTag (Page 8032)

ILoggedTagValueEnumerator

Description

The "ILoggedTagValueEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of process values of a logging tag of the Runtime system. The methods are used by the C++-interfaces "ILoggedTag" and "ILoggedTagSet".

All the methods return CF_SUCCESS in case of successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(ILoggedTagValue **ppItem)
```

`ppItem`
[out]: Points to the current "ILoggedTagValue" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext"method subsequently moves to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of the elements of the list.

Example

Output process values of a logging tag:

Copy code

```
void PrintValues(ILoggedTagValueEnumeratorPtr pItems,IRuntimePtr pRuntime)
{
   // Iterate over the process value objects
   while (CF_SUCCEEDED(pItems->MoveNext()))
   {
      ILoggedTagValuePtr pValue;
      pItems->Current(&pValue);   // get current process value
      ...
   }
}
```

See also

ILoggedTagValue (Page 8024)

ILoggedTag (Page 8032)

ILoggedTagSet (Page 8033)

ILoggedTagCallback / ILoggedTagSetCallback (Page 8028)

ILoggedTagCallback / ILoggedTagSetCallback

Description

The interfaces "ILoggedTagCallback" and "ILoggedTagSetCallback" and the classes "COdkTagSourceCBBase" and "COdkTagSetLoggingCB" define methods for implementing asynchronous read and write operations with logging tags. The methods are used by the C++-interfaces "ILoggedTag" and "ILoggedTagSet".

All the methods return CF_SUCCESS following successful execution.

Members of "ILoggedTagCallback"

The following methods are specified in the interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

```
CFRESULT OnReadComplete(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

"OnDeleteComplete" method

Callback method is called on completion of asynchronous delete operations.

```
CFRESULT OnDeleteComplete(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

"OnWriteComplete" method

Callback method is called on completion of asynchronous write operations. Can only be applied to individual logging tags in the "ILoggedTagCallback" interface.

```
CFRESULT OnWriteComplete(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

"OnDataChanged" method

Callback method is called when a monitored logging tag is changed.

```
CFRESULT OnDataChanged(ILoggedTagValueEnumerator *pEnumerator)
```

`pEnumerator`
[out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.

Members of "ILoggedTagSetCallback"

The following methods are specified in the interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations.

The "OnReadComplete" callback method is called when the "ReadAsync" method is used.

```
CFRESULT OnReadComplete(ILoggedTagValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logging tag.

"OnDataChanged" method

Callback method is called when a monitored logging tag is changed.

The callback method is called after the process value change of a monitored logging tag or a LoggedTagSet.

```
CFRESULT OnDataChanged(ILoggedTagValueEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegTagValueEnumerator" object that contains the enumeration of the process values.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logging tag.

Members of the classes

The following methods are implemented in the "COdkTagSourceCBBase" and "COdkTagSetLoggingCB" classes:

"SetEvent" method

Signals an event.

```
CFBOOL SetEvent()
```

"ResetEvent" method

Resets the signaling of an event.

```
CFBOOL ResetEvent()
```

"WaitForcompletion" method

Waits for the signaling of an event.

```
uint32_t WaitForcompletion(uint32_t dwMilliseconds)
```

`dwMilliseconds`
[in]: Time interval in milliseconds for which an event is waited for.

"GetValues" method

Return process values of the asynchronous read operation.

```
std::vector<ILoggedTagValuePtr> GetValues()
```

Example

Output LoggedTagSet asynchronously:

Copy code

```
void LogggingReadTagSetAsync(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"LoggedTagSet"), &pUnk);

   ILoggedTagSetPtr pTagSet(pUnk);
   // add tags to tag set
   for (int i = 0; i < tags.size(); i++)
   {
      pTagSet->Add(tags[i]);
   }
   CCfDateTime64 begin, end;
   begin = CCfDateTime64::Now(true);
   end = begin;
   begin.SubtractTimeSpan(Get1Minute() * 30);

   ILoggedTagValueEnumeratorPtr pItems;

   COdkTagSetLoggingCB* pTagSetCB = new COdkTagSetLoggingCB();

   pTagSetCB->AddRef();
   // Read the tag set asynchronously, result comes via callback
   pTagSet->ReadAsync(pTagSetCB, begin, end, true);
   pTagSetCB->WaitForcompletion(std::numeric_limits<uint32_t>::max());
   vector<ILoggedTagValuePtr> pValues = pTagSetCB->GetValues();

   // display tag values
   for (int i = 0; i < pValues.size(); i++)
   {
      ILoggedTagValue* pValue = pValues[i];
      CCfString timeStamp;
      CFDATETIME64 cfTimeStamp;
      pValue->GetTimeStamp(&cfTimeStamp);
      CCfDateTime64 time(cfTimeStamp);
      timeStamp = time.GetDateTimeString(false);
      CCfString strName;
      pValue->GetTagName(&strName);
      CCfVariant varValue;
      pValue->GetValue(&varValue);
      std::wcout << strName.ToUTF8().c_str() << L" " << timeStamp.ToUTF8().c_str() << L"
Value: " << (double)(varValue) << std::endl;
   }
}
```

See also

ILoggedTag (Page 8032)

ILoggedTagSet (Page 8033)

ILoggedTagValueEnumerator (Page 8027)

ILoggedTag

Description

The C++ interface "ILoggedTag" specifies properties and methods for the handling of logging tags of a logging system. A logging tag is represented by an "ILoggedTag" instance. The information on the logged process values of the logging tag is stored in "ILoggedTagValue" instances.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetTagName" method

Set name of the logging tag.

```
CFRESULT SetTagName(const CFSTR tagName)
```

`tagName`
[in]: Name of the logging tag

"Read" method

Synchronous read operation to read the "ILoggedTagValue" instances of the logging tag whose time stamps are in the time period defined by the arguments.

```
CFRESULT Read(
         const CFTIMEDATE64 begin,
         const CFTIMEDATE64 end,
         ILoggedTagValueEnumerator** ppEnumerator,
         CFBOOL boundingValue)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `ppEnumerator`
  [in/out]: Points to the enumeration of "ILoggedTagValue" instances of the logging tag as "ILoggedTagValueEnumerator" object.
- `boundingValue`
  [in]: True, in order to additionally return high and low limits.

"Write" method

Synchronous write operation.

The method manually writes an "ILoggedTagValue" instance to the logging system. "Quality" of the instance receives the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

Application example: While the connection to the logging system is interrupted, the process value of a tag that is being logged changes. The resulting "ILoggedTagValue" instance is cached. After the connection has been re-established, manually add the instance to the log by calling "Write".

```
CFRESULT Write(
         ILoggedTagValue* ptag
```

- `ptag`
  [in]: The "ILoggedTagValue" instance to be logged

Example

Read out process values of a logging tag synchronously from a logging system:

Copy code

```
void LoggingReadSingleTagSync(IRuntimePtr pRuntime, CCfString tag)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"LoggedTag"), &pUnk);
   ILoggedTagPtr pTag(pUnk);
   pTag->SetTagName(tag);
   CCfDateTime64 begin, end;
   begin = CCfDateTime64::Now(true);
   end = begin;
   begin.SubtractTimeSpan(Get1Minute() * 3);
   ILoggedTagValueEnumeratorPtr pItems;
   // Read value of tag
   pTag->Read(begin, end, &pItems, true);
   std::wcout << "Read finished " << std::endl;
   PrintValues(pItems ,pRuntime);
}
```

See also

ILoggedTagValue (Page 8024)

ILoggedTagSet (Page 8033)

ILoggedTagCallback / ILoggedTagSetCallback (Page 8028)

ILoggedTagValueEnumerator (Page 8027)

ILoggedTagSet

Description

The C++ interface "ILoggedTagSet" specifies properties, methods and events for optimized access to a collection of "ILoggedTag" instances of a logging system.

After initializing an "ILoggedTagSet" instance, you can read or write multiple "ILoggedTag" instances in one call. Simultaneous access demonstrates better performance and lower communication load than single access to multiple logging tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetContextId" method

ID as additional identification feature of the logging tag. The ContextId can, for example, be used to recognize identically named logging tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT SetContextId(const int32_t value)
```

`value`
[in]: ContextId of the logging tag

"GetContextId" method

ID as additional identification feature of the logging tag. The ContextId can, for example, be used to recognize identically named logging tags from different monitoring functions. Default value -1: The ContextId is not used.

```
CFRESULT GetContextId(int32_t* value)
```

`value`
[out]: Points to the ContextId of the logging tag.

"Add" method

The method is overloaded:

- Add a single "ILoggedTag" instance to the "ILoggedTagSet" instance:
  `CFRESULT Add(const CFSTR tagName)`
  `tagName`
  [in]: Name of the "ILoggedTag" instance
- Add an "ILoggedTagValue" instance to the "ILoggedTagSet" instance:
  `CFRESULT Add(ILoggedTagValue* object)`
  `object`
  [in]: Reference of the "ILoggedTagValue" instance

"Remove" method

Remove individual logging tag from a LoggedTagSet.

```
CFRESULT Remove(const CFSTR tagName)
```

`tagName`
[in]: Name of the logging tag that is removed from the LoggedTagSet.

"Clear" method

Remove all logging tags from a LoggedTagSet.

```
CFRESULT Clear()
```

"Read" method

Retrieve all logging tags of a LoggedTagSet for a period of time synchronously from the logging system.

```
CFRESULT Read(
         const CFTIMEDATE64 begin,
         const CFTIMEDATE64 end,
         ILoggedTagValueEnumerator** ppEnumerator,
         CFBOOL boundingValue)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `ppEnumerator`
  [in/out]: Points to the enumeration of process values of the logging tags of the LoggedTagSet as "ILoggedTagValueEnumerator" object.
- `boundingValue`
  [in/optional]: True, in order to additionally return high and low limits.

"ReadAsync" method

Retrieve all logging tags of a LoggedTagSet for a period of time synchronously from the logging system.

```
CFRESULT ReadAsync(
         ILoggedTagSetCallback *pTagLoggedCb,
         const CFTIMEDATE64 begin,
         const CFTIMEDATE64 end,
         CFBOOL boundingValue)
```

- `pTagLoggedCb`
  [in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.
- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `boundingValue`
  [in/optional]: True, in order to additionally return high and low limits.

"Subscribe" method

Subscribe all logging tags of a LoggedTagSet asynchronously for updating the process values following a change. When new process values are logged, they can be processed with the "OnDataChanged" event.

```
CFRESULT Subscribe(ILoggedTagSetCallback* pTagSetLoggedCb)
```

`pTagSetLoggedCb`
[in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.

"CancelSubscribe" method

Cancel updating of process values following a change for all logging tags of a LoggedTagSet.

```
CFRESULT CancelSubscribe()
```

"Write" method

Synchronous write operation.

Writes the "ILoggingTagValue" instances added by call of "Add" to the logging system. "Quality" of the instance receives the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

Errors during write operations are returned in a list with "IErrorResult" instances.

Application example: The connection to the archive server was interrupted. After the connection has been re-established, add multiple "ILoggedTagValue" instances cached in the PLC to the archive afterwards by calling "Write".

```
CFRESULT Write(IErrorResultEnumerator** ppErrorEnumerator)
```

`ppErrorEnumerator`
[out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations.

"WriteAsync" method

Asynchronous write operation.

Writes the "ILoggingTagValue" instances added by call of "Add" to the logging system. "Quality" of the instance receives the value "GOOD" (sub-status: "Unspecific", extended sub-status: "Manual input"). The "Source time" marker is set to "1".

Application example: The connection to the archive server was interrupted. After the connection has been re-established, add multiple "ILoggedTagValue" instances cached in the PLC to the archive afterwards by calling "Write".

```
CFRESULT WriteAsync(ILoggedTagSetCallback* pTagLoggedCb)
```

`pTagLoggedCb`
[in]: Points to the "ILoggedTagSetCallback" object that implements the callback interface.

"GetCount" method

Return the number of logging tags of a LoggedTagSet list.

```
CFRESULT GetCount(int32_t *value)
```

`value`
[out]: Points to the value for the number of logging tags of the LoggedTagSet list.

Example

Subscribe logging tags of a LoggedTagSet for change monitoring:

Copy code

```
void LoggingSubscribeTagSet(IRuntimePtr pRuntime, std::vector<CCfString> tags)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"LoggedTagSet"), &pUnk);
   ILoggedTagSetPtr pTagSet(pUnk);
   // add tags to tag set
   for (int i = 0; i < tags.size(); i++)
   {
      pTagSet->Add(tags[i]);
   }
   COdkTagSetLoggingCB* pTagSetCB = new COdkTagSetLoggingCB();
   pTagSetCB->AddRef();
   // subscribe tags
   pTagSet->Subscribe(pTagSetCB);
   //stop subscription
   pTagSet->CancelSubscribe();
}
```

See also

ILoggedTag (Page 8032)

ILoggedTagCallback / ILoggedTagSetCallback (Page 8028)

ILoggedTagValueEnumerator (Page 8027)

ITags

Description

The C++ interface "ITags" defines methods with which you can access configured tags.

The interface inherits from the "ICfDispatch" interface.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"Find" method

Supplies an enumerator for access to the "ITagAttributes" instances of the specified tag.

```
CFRESULT Find(
         CFVARIANT systemIDs,
         int32_t language,
         CFSTR filter,
         ITagAttributesEnumerator** ppITagAttributesEnumerator)
```

```
Find(CFVARIANT systemIDs, uint32_t language, CFSTR filter,
ITagAttributesEnumerator** ppITagAttributesEnumerator)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*)
- `ppITagAttributesEnumerator`
  [out]: Enumerator which supplies access to the "ITagAttributes" instances.

"FindAsync" method

For asynchronous searching for "ITagAttributes" instances of the specified tags.

```
FindAsync(
         CFVARIANT systemIDs
         uint32_t language
         CFSTR filter
         ITagAttributesCallback* pCallback)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the tags to restrict the search.
  Supports searching with wildcard (*)
- `filterpCallback`:
  [in]: Callback pointer to an "ITagAttributesCallback" instance

ITagAttributes

Description

The C++ interface "ITagAttributes" defines methods for access to the attributes of a tag.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the tags. Must be unique throughout the device.

```
GetName(
         CFSTR* name)
```

- `name`
  [out]: The name

"GetDisplayName" method

Supplies the display name of the tags.

```
GetDisplayName(
         CFSTR* name)
```

- `name`
  [out]: The display name

"GetDataType" method

Supplies the data type of the tags.

```
GetDataType(
         CFENUM* datatype)
```

- `datatype`
  [out]: The data type of the tag

"GetConnection" method

Supplies the connection of the tag.

The memory location of the tag in the controller is accessed via the connection.

```
GetConnection(
         CFSTR* connection)
```

- `connection`
  [out]: The connection

"GetAcquisitionCycle" method

Specifies the tag acquisition cycle.

If you configure a tag at an object, the acquisition cycle of the tag is displayed at the object.

```
GetAcquisitionCycle(
         CFSTR* acquisitionCycle)
```

- `acquisitionCycle`
  [out]: The acquisition cycle

"GetAcquisitionMode" method

Supplies the acquisition mode of the tag.

```
GetAcquisitionMode(
         CFENUM* acquisitionMode)
```

- `acquisitionMode`
  [out]: Value of the enumeration "HmiAcquisitionMode".
  The enumeration "HmiAcquisitionMode" can contain the following values:
  - Undefined (0)
  - CyclicOnUse (1)
  - CyclicContinous (2)
  - OnDemand (3)
  - OnChange (4)

"GetMaxLength" method

Supplies the length of the tags.

The length is only available with a string tag. The length is preset for structure tags and cannot be changed.

```
GetMaxLength(
         uint32_t* maxLength)
```

- `maxLength`
  [out]: Maximum length

"GetPersistent" method

Supplies the persistence of the tags.

```
GetPersistent(
         CFBOOL* persistent)
```

- `persistent`
  [out]: The persistence

"GetInitialValue" method

Supplies the start value of the tag.

After Runtime starts, the tag retains the start value until an operator or the PLC changes the value.

```
GetInitialValue(
        CFVARIANT * initialValue)
```

- `initialValue`
  [out]: The start value

"GetInitialMaxValue" method

Supplies the start value for the event "On exceeding".

```
InitialMaxValue(
        CFVARIANT * initialValue)
```

- `initialValue`
  [out]: The start value

"GetInitialMinValue" method

Supplies the start value for the event "On falling below".

```
InitialMinValue(
        CFVARIANT * initialValue)
```

- `initialValue`
  [out]: The start value

"GetSubstituteValue" method

Supplies the substitute value of the tags.

If the selected condition occurs, the tag is filled with the substitute value in runtime.

```
GetSubstituteValue(
        CFVARIANT * substituteValue)
```

- `substituteValue`
  [out]: The substitute value

"GetSubstituteValueUsage" method

Supplies the condition for the use of the substitute value of the tag.

```
GetSubstituteValueUsage(
        CFVARIANT * substituteValueUsage)
```

- `substituteValueUsage`
  [out]: The condition

ITagAttributesEnumerator

Description

The "ITagAttributesEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of tag attributes. The enumerator enables access to individual attributes from a set of tag attributes.

The interface inherits from the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumerator.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumerator

```
CFRESULT Current(
         ITagAttributes** ppItem)
```

- `ppItem`
  [out]: The current "ITagAttributes" instance

"Reset" method

Reset the current position in the enumerator. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumerator or the number of its elements.

```
CFRESULT Count(
         uint32_t* value)
```

- `value`
  [out]: Number of attributes

ITagAttributesCallback

Description

The C++ interface "ITagAttributesCallback" defines the callback method "OnTagAttributesRead". The method is used for implementing asynchronous read operations of tag attributes. The method is used by the C++ interface "ITags".

The interface implements the methods of the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnTagAttributesRead" method

Callback method, is called on completion of an asynchronous search for "ITagAttributes" instances.

```
OnTagAttributesRead(
         ITagAttributesEnumerator* pEnumerator,
         CFBOOL bIsCompleted)
```

- `ITagAttributesEnumerator`
  [in/out]: Reference to an "ITagAttributesEnumerator" instance which provides access to the tag attributes.
- `bIsCompleted`
  [out]: Supplies information on whether the read operation was successfully completed.

ILoggingTags

Description

The C++ interface "ILoggingTags" defines methods with which you can access configured logging tags.

The interface inherits from the "ICfDispatch" interface.

All the methods return `CF_SUCCESS` following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"Find" method

Supplies an enumerator for access to the "ITagAttributes" instances of the specified logging tag.

```
CFRESULT Find(
         CFVARIANT systemIDs,
         uint32_t language,
         CFSTR filter,
        ILoggingTagAttributesEnumerator** ppILoggingTagAttributesEnumerator)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the logging tags were configured.
- `language`
  [in]: Language code ID of filter
- `filter`
  [in]: Filter by name of the logging tags to restrict the search.
  Supports searching with wildcard (*).
  Example:
  `Tag1:*` Supplies all logging tags of "Tag1".
- `ppILoggingTagAttributesEnumerator`
  [out]: Enumerator which supplies access to the "ILoggingTagAttributes" instances.

"FindAsync" method

For asynchronous searching for "ILoggingTagAttributes" instances.

```
CFRESULT FindAsync(CFVARIANT systemIDs,
         uint32_t language,
         CFSTR filter,
         ILoggingTagAttributesCallback* pCallback)
```

- `systemIDs`
  [in]: Collection of SystemNames on which the logging tags were configured.
- `language`
  [in]: Language code ID of filter

- `filter`
  [in]: Filter by name of the logging tags to restrict the search.
  Supports searching with wildcard (*).
- pCallback
  [in]: Callback pointer to an "ILoggingTagAttributesCallback" instance

ILoggingTagAttributes

Description

The C++ interface "ITagAttributes" defines methods for access to the attributes of a logging tag.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the logging tag.

```
GetName(
         CFSTR* name)
```

- `name`
  [out]: The name

"GetDisplayName" method

Supplies the display name of the logging tags.

```
GetDisplayName(
         CFSTR* name)
```

- `name`
  [out]: The display name

"GetDataType" method

Supplies the data type of the logging tags.

```
GetDataType(
         CFENUM* datatype)
```

- `datatype`
  [out]: The data type

ILoggingTagAttributesEnumerator

Description

The "ILoggingTagAttributesEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of logging tag attributes. The enumerator enables access to individual attributes from a set of tag attributes.

The interface inherits from the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumerator.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumerator

```
CFRESULT Current(
         ITagAttributes** ppItem)
```

- `ppItem`
  [out]: The current "ITagAttributes" instance

"Reset" method

Reset the current position in the enumerator. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumerator or the number of its elements.

```
CFRESULT Count(
         uint32_t* value)
```

- `value`
  [out]: Number of attributes

ILoggingTagAttributesCallback

Description

The C++ interface "ILoggingTagAttributesCallback" defines the callback method "OnTagAttributesRead". The method is used for implementing asynchronous read operations of logging tag attributes. The method is used by the C++ interface "ILoggingTags".

The interface implements the methods of the "ICfUnknown" interface.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnTagAttributesRead" method

Callback method, is called on completion of an asynchronous search for "ITagAttributes" instances.

```
OnTagAttributesRead(
          ILoggingTagAttributesEnumerator* pEnumerator,
          CFBOOL bIsCompleted)
```

- `ILoggingTagAttributesEnumerator`
  [in]: Reference to an "ILoggingTagAttributesEnumerator" instance which provides access to the tag attributes.
- `bIsCompleted`
  Supplies information on whether the result of the read operation is complete or whether other events will come.

20.3.9.4 Interfaces of the alarms

IAlarmResult

Description

The C++ interface "IAlarmResult" specifies methods for accessing properties of active alarms of the runtime system.

An "IAlarmResult" object is a pure data object which maps all properties of an active alarm.

Members

The following methods are specified in the interface:

"GetInstanceID" method

Return InstanceID for an alarm with multiple instances.

```
CFRESULT GetInstanceID(uint32_t *value)
```

`value`
[out]: Points to the InstanceID of the alarm.

"GetSourceID" method

Return source at which the alarm was triggered.

```
CFRESULT GetSourceID(CFSTR *value)
```

`value`
[out]: Points to the source of the alarm.

"GetName" method

Return the name of the alarm.

```
CFRESULT GetName(CFSTR *value)
```

`value`
[out]: Points to the name of the alarm.

"GetAlarmClassName" method

Return the name of the alarm class.

```
CFRESULT GetAlarmClassName(CFSTR *value)
```

`value`
[out]: Points to the symbol of the associated alarm class.

"GetAlarmClassSymbol" method

Return the symbol of the alarm class.

```
CFRESULT GetAlarmClassSymbol(CFSTR *value)
```

`value`
[out]: Points to the name of the associated alarm class.

"GetState" method

Return current alarm state.

```
CFRESULT GetState(int32_t *value)
```

`value`
[out]: Points to the current alarm state.

"GetStateText" method

Return current alarm state as text, for example, "active" or "inactive".

```
CFRESULT GetStateText(CFSTR *value)
```

`value`
[out]: Points to the current alarm state as text.

"GetEventText" method

Return text that describes the alarm event.

```
CFRESULT GetEventText(CFSTR *value)
```

`value`
[out]: Points to the text that describes the alarm event.

"GetInfoText" method

Return text that describes the alarm event.

```
CFRESULT GetInfoText(CFSTR *value)
```

`value`
[out]: Points to the text that describes an operator instruction for the alarm.

"GetAlarmText1" ... "GetAlarmText9" method

Return additional texts 1-9 of the alarm.

```
CFRESULT GetAlarmText1(CFSTR *value)
```

...

```
CFRESULT GetAlarmText9(CFSTR *value)
```

`value`
[out]: Points to the additional text of the alarm.

"GetTextColor" method

Return text color of the alarm state.

```
CFRESULT GetTextColor(uint32_t *value)
```

`value`
[out]: Points to the text color of the alarm state.

"GetBackColor" method

Return background color of the alarm state.

```
CFRESULT GetBackColor(uint32_t *value)
```

`value`
[out]: Points to the background color of the alarm state.

"GetFlashing" method

Return flashing background color of the alarm state.

```
CFRESULT GetFlashing(CFBOOL *value)
```

`value`
[out]: Points to the flashing background color of the alarm state.

"GetModificationTime" method

Return time of the last modification to the alarm state.

```
CFRESULT GetModificationTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the last change of the alarm state.

"GetChangeReason" method

Return trigger event for modification of the alarm state.

```
CFRESULT GetChangeReason(uint16_t *value)
```

`value`
[out]: Points to the trigger event for the change of the alarm state.

"GetRaiseTime" method

Return trigger time of the alarm.

```
CFRESULT GetRaiseTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time the alarm was triggered.

"GetAcknowledgementTime" method

Return time of alarm acknowledgment.

```
CFRESULT GetAcknowledgementTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of alarm acknowledgment.

"GetClearTime" method

Return time of the alarm reset.

```
CFRESULT GetClearTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the alarm reset.

"GetResetTime" method

Return time of the alarm reset.

```
CFRESULT GetResetTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the alarm reset.

"GetSuppressionState" method

Return status of alarm visibility.

```
CFRESULT GetSuppressionState(uint8_t *value)
```

`value`
[out]: Points to the status of the alarm visibility.

"GetPriority" method

Return relevance for display and sorting of the alarm.

```
CFRESULT GetPriority(uint8_t *value)
```

`value`
[out]: Points to the relevance of the alarm.

"GetOrigin" method

Return origin for display and sorting of the alarm.

```
CFRESULT GetOrigin(CFSTR *value)
```

`value`
[out]: Points to the origin of the alarm.

"GetArea" method

Return origin area for display and sorting of the alarm.

```
CFRESULT GetArea(CFSTR *value)
```

`value`
[out]: Points to the origin area of the alarm.

"GetValue" method

Return the current process value of the alarm.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the current process value of the alarm.

"GetValueQuality" method

Return quality of the process value of the alarm.

```
CFRESULT GetValueQuality(uint16_t *value)
```

`value`
[out]: Points to the quality of the process value of the alarm.

"GetValueLimit" method

Return process value limit of the alarm.

```
CFRESULT GetValueLimit(CFVARIANT *value)
```

`value`
[out]: Points to the limit of the process value of the alarm.

"GetUserName" method

Return the user name of the operator input alarm.

```
CFRESULT GetUserName(CFSTR *value)
```

`value`
[out]: Points to the user name of the operator input alarm.

"GetLoopInAlarm" method

Return function that navigates from the alarm control to its origin.

```
CFRESULT GetLoopInAlarm(CFSTR *value)
```

`value`
[out]: Points to the function name that navigates to the origin of the alarm.

"GetAlarmParameterValues" method

Return parameter values of the alarm.

```
CFRESULT GetAlarmParameterValues(CFVARIANT *value)
```

`value`
[out]: Points to the parameter values of the alarm.

"GetInvalidFlags" method

Return identification of the alarm with invalid data.

```
CFRESULT GetInvalidFlags(uint8_t *value)
```

`value`
[out]: Points to the invalid data of the alarm.

"GetConnection" method

Return connection via which the alarm was triggered.

```
CFRESULT GetConnection CFSTR *value)
```

`value`
[out]: Points to the connection of the alarm.

"GetSystemSeverity" method

Return severity of the system error.

```
CFRESULT GetSystemSeverity(uint8_t *value)
```

`value`
[out]: Points to the severity of the system error.

"GetUserResponse" method

Return expected or required user response to the alarm.

```
CFRESULT GetUserResponse(uint8_t *value)
```

`value`
[out]: Points to the expected or required user response to the alarm.

"GetSourceType" method

Return source from which the alarm was generated, for example, tag-based, controller-based or system-based alarm.

```
CFRESULT GetSourceType(uint16_t *value)
```

`value`
[out]: Points to the type of source.

"GetDeadBand" method

Return range of the triggering tag in which no alarms are generated.

```
CFRESULT GetDeadBand(CFVARIANT *value)
```

`value`
[out]: Points to the non-triggering range.

"GetId" method

Return ID of the alarm that is also used in the display.

```
CFRESULT GetId(uint32_t *value)
```

`value`
[out]: Points to the ID of the alarm.

"GetAlarmGroupID" method

Return ID of the alarm group to which the alarm belongs.

```
CFRESULT GetAlarmGroupID(uint32_t* groupId)
```

`groupId`
[out]: The ID of the alarm group

"GetHostName" method

Return the name of the host that triggered the alarm.

```
CFRESULT GetHostName(CFSTR *value)
```

`value`
[out]: Points to the host name.

"GetNotificationReason" method

Return the reason for the notification.

```
CFRESULT GetNotificationReason(CFENUM* value)
```

`value`:
[out]: Points to the enumeration of the notification reason. The notification reason can have the following values:

- `Unknown` (0)
- `Add` (1)
  The alarm was added to the filtered result list. The alarm meets the filter criteria that apply to the monitoring.
- `Modify` (2)
  Properties of the alarm were changed, but the alarm is still part of the filtered result list.
- `Remove` (3)
  The alarm was part of the result list, but it no longer meets the filter criteria due to changes to its properties.

  Note

  Changes to the alarms only lead to notifications again when the alarm meets the filter criteria again. In this case, "NotificationReason" is set to `Add`.

Note

Removing alarm from business logic

The use case of the client determines whether the client ignores notifications via alarms with the "NotificationReason" `Modify` or `Remove`.

For example:

- State-based monitoring: The client wants to show a list of incoming alarms. All notification reasons are relevant. The client removes an alarm from the list as soon as the notification reason is `Remove`.
- Event-based monitoring: The client wants to send an email when an alarm comes in. Only the notification reason `Add` is relevant.

Example:

An ODK client begins monitoring with the filter criterion "State" = 1. An alarm is triggered. Runtime notifies the ODK client of the "NotificationReason" as follows:

Notification-Reason	Description
Add	• The "State" property is 1. The alarm is active.
Modify	• The "State" property has not changed. • Another property that is not part of the filter criterion has changed, e.g. "Priority".
Remove	The "State" property has changed, for example, alarm is inactive.

Method "GetDuration"

Returns the time interval in nanoseconds between triggering of the alarm and its previous status change.

```
CFRESULT GetDuration(CFTIMESPAN64 *value)
```

`value`
[out]: The time interval in nanoseconds

Example

When alarms become active, a selection of properties of the "IAlarmResult" objects is output:

Copy code

```
// Callback for alarm notifications
CFRESULT CFCALLTYPE CAlarmValue::OnAlarm(IAlarmResultEnumerator* pItems, uint32_t
systemError, CFSTR systemName, CFBOOL completed)
{
    CFRESULT hr = CF_FALSE;
    CFBOOL bSet = false;
    uint32_t nsize;
    pItems->Count(&nsize);

    if (nsize > 0 && CF_SUCCEEDED(systemError)) {
        m_AlarmValue.clear();

        while (CF_SUCCEEDED(pItems->MoveNext()))
        {
            IAlarmResultPtr ppValues;
            pItems->Current(&ppValues);

            AlarmAttributes AlarmValue;
            AlarmValue.m_nInstanceID;
            ppValues->GetInstanceID(&AlarmValue.m_nInstanceID);
            AlarmValue.m_strSourceID;
            CCfString strId;
            ppValues->GetSourceID(&strId);
            AlarmValue.m_strSourceID = CCfSmartString(strId);
            CCfString strName;
            ppValues->GetName(&strName);
            AlarmValue.m_strName = CCfSmartString(strName);
            CCfString strClassName;
            ppValues->GetAlarmClassName(&strClassName);
            AlarmValue.m_strAlarmClassName = CCfSmartString(strClassName);
            ppValues->GetState(&AlarmValue.m_nState);
            CCfString strEvent;
            ppValues->GetEventText(&strEvent);
            AlarmValue.m_strEventText = CCfSmartString(strEvent);
            CCfString strText;
            ppValues->GetStateText(&strText);
            AlarmValue.m_strStateText = CCfSmartString(strText);
            ppValues->GetBackColor(&AlarmValue.m_nBackColor);
            ppValues->GetTextColor(&AlarmValue.m_nTextColor);
            ppValues->GetFlashing(&AlarmValue.m_bFlashing);
            ppValues->GetDuration(&nDuration);

            ...

            AlarmValue.m_nAlarmsSize = nsize;

            AlarmValue.m_strSystemName = systemName;
           std::cout << "System name = " << AlarmValue.m_strSystemName.ToUTF8().c_str() <<
std::endl;

            m_AlarmValue.push_back(AlarmValue);
            std::cout << "Alarm name = " << strName.ToUTF8().c_str() << std::endl;
        }

        this->SetEvent();
```

Copy code

```
        return CF_SUCCESS;
    }
    return hr;
}
```

See also

IAlarmResultEnumerator (Page 8056)

IAlarm (Page 8057)

IAlarmSubscription (Page 8077)

IAlarmResultEnumerator

Description

The "IAlarmResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of active alarms of the runtime system. Through the enumeration, you access individual alarms from the quantity of all active alarms of the runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IAlarmResult **ppItem)
```

`ppItem`
[out]: Points to the current "IAlarmResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

See also

IAlarmResult (Page 8046)

IAlarm

Description

The C++ interface "IAlarm" specifies properties and methods for handling active alarms of the runtime system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetSourceID" method

Set the ID of the source at which the active alarm was triggered.

```
CFRESULT SetSourceID(CFVARIANT sourceID)
```

`sourceID`
[in]: ID of the source of the active alarm

"SetName" method

Set the name of the active alarm.

```
CFRESULT SetName(CFSTR name)
```

`name`
[in]: Name of the active alarm

"GetName" method

Read out name of the active alarm.

```
CFRESULT GetName(CFSTR *name)
```

`name`
[out]: Points to a name of the active alarm.

"SetErrorCode" method

Set error code of the alarm.

```
CFRESULT SetErrorCode(uint32_t errorCode)
```

`errorCode`
[in]: Error code

"GetError" method

Read out error code of the alarm.

```
CFRESULT GetError(uint32_t *error)
```

`error`
[out]: Error code

"Disable" method

Deactivate generation of the alarm in the alarm source synchronously.

```
CFRESULT Disable()
```

"Enable" method

Activate generation of the alarm in the alarm source again synchronously.

```
CFRESULT Enable()
```

"Shelve" method

Hide active alarm synchronously.

```
CFRESULT Shelve()
```

"Unshelve" method

Display hidden alarm synchronously again.

```
CFRESULT Unshelve()
```

"Acknowledge" method

Acknowledge active alarm or instance of an active alarm synchronously.

```
CFRESULT Acknowledge(uint32_t instanceId)
```

- `instanceId`
  Value "0": Acknowledge all instances of an active alarm.
  Value > "0": Acknowledge instance with this ID.

"Reset" method

Acknowledge the inactive state of an active alarm or an instance of an active alarm synchronously.

```
CFRESULT Reset(uint32_t instanceId)
```

- `instanceId`
  Value "0": Acknowledge the inactive state of the active alarm.
  Value > "0": Acknowledge the inactive state of an instance with this ID.

Example

Acknowledge list of active alarms synchronously:

Copy code

```
vector<AlarmAttributes> g_vecAlarmList;

void AcknowledgeAlarmSync(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    pRuntime->GetObject(CCfString(L"Alarm"), &pUnk);

      IAlarmPtr pAlarm(pUnk);
      pAlarm->SetName("Tag1:Alarm1");
      CFRESULT errCode = pAlarm->Acknowledge();
    }
}
```

See also

IAlarmResult (Page 8046)

IAlarmCallback

Description

The C++ interface "IAlarmCallback" defines methods for implementing asynchronous operations for monitoring active alarms. The methods are used by the "IAlarmSubscription" interface.

Members

The following methods are specified in the "IAlarmCallback" interface:

"OnAlarm" method

Callback method is called when active alarms on monitored systems change the alarm state.

The "OnAlarm" callback method is called when the "IAlarmSubscription.Start" method is used.

```
CFRESULT OnAlarm(
         IAlarmResultEnumerator* pItems,
         uint32_t systemError,
         CFSTR systemName,
         CFBOOL completed)
```

- `pItems`
  [in]: Points to an "IAlarmResultEnumerator" object that contains the enumeration of the changed active alarms.
- `systemError`
  [in]: Error code for the asynchronous operation

- `systemName`
  [in]: System of the associated Runtime system.
- `completed`
  [in]: True, if no more data from the callback can be expected.

"OnPendingAlarmsComplete" method

Callback method is called to display the completion of the handling of all the active alarms of a system.

The "OnPendingAlarmsComplete" callback method is called when the "IAlarmSubscription.Start" method is used.

```
CFRESULT OnPendingAlarmsComplete()
```

Example

In the following section, monitored active alarms are written asynchronously into the "g_vecAlarmList" map at a change. To this purpose the "SubscribeAlarm" function uses a "CAlarmValue" object that implements the "IAlarmCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CAlarmValue" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void SubscribeAlarm(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk));
   IAlarmSubscriptionPtr pAlarm(pUnk);
   CAlarmValue* pAlarmValue = new CAlarmValue();
   pAlarmValue->AddRef();
   IAlarmCallback* pCB = pAlarmValue;
   CCfSafeArrayBound bounds(1UL, 0);
   CCfSafeArray daAttribute;
   CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
   CCfVariant daDataSource = 0;
   CCfVariant vSystemIDs = 0;
   // Subscription filter is currently not supported!
   CCfSmartString daFilter = L"";
   CCfSREF id(L"SYSTEM1");
   int32_t index = 0;
   daSystemID.PutElement(&index, &id);
   CCfVariant daLanguage = 1033;

   daSystemID.Detach(&vSystemIDs);
   CCfSmartString strFilter = L"";
   // Start Subscription
   pAlarm->SetFilter(strFilter.AllocCFSTR());
   pAlarm->SetLanguage(1033);
   pAlarm->SetSystemNames(vSystemIDs);
   Alarm->Start(pCB);
   // Wait for alarm notifications
   pAlarmValue->WaitForcompletion(g_nMaxWaitTime);
   Alarm->Stop();
   pAlarmValue->GetAlarmAttributes(g_vecAlarmList);
}
```

See also

IAlarmSubscription (Page 8077)

IAlarmSourceCommandCallback

Description

The C++ interface "IAlarmSourceCommandCallback" defines methods for implementing asynchronous operations with active alarms. The methods are used by the "IAlarmSet" interface.

Members

The following methods are specified in the interface:

"OnAcknowledge" method

Callback method is called when an active alarm has been acknowledged.

The "OnAcknowledge" callback method is called when the "IAlarmSet.Acknowledge" method is used.

```
CFRESULT OnAcknowledge(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnReset" method

Callback method is called when an active alarm has been removed.

The "OnReset" callback method is called when the "IAlarmSet.Reset" method is used.

```
CFRESULT OnReset(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnDisable" method

Callback method is called when the generation of the alarm in the alarm source has been deactivated.

The "OnDisable" callback method is called when the "IAlarmSet.Disable" method is used.

```
CFRESULT OnDisable(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnEnable" method

Callback method is called when the generation of the alarm in the alarm source has been activated again.

The "OnEnable" callback method is called when the "IAlarmSet.Enable" method is used.

```
CFRESULT OnEnable(
         CFRESULT SystemError,
         IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnShelve" method

Callback method is called when an active alarm has been hidden.

The "OnShelve" callback method is called when the "IAlarmSet.Shelve" method is used.

```
CFRESULT OnShelve(
         CFRESULT SystemError,
```

```
        IAlarmSetResultEnumerator *AlarmSetResult,
        CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

"OnUnShelve" method

Callback method is called when an active alarm has been shown.

The "OnUnshelve" callback method is called when the "IAlarmSet.Unshelve" method is used.

```
CFRESULT OnUnshelve(
         CFRESULT SystemError,
        IAlarmSetResultEnumerator *AlarmSetResult,
         CFBOOL MoreFollows)
```

- `SystemError`
  [in]: Basic errors that occurred during the asynchronous transfer.
- `AlarmSetResult`
  [in]: Points to an enumeration with the alarms of the callback
- `MoreFollows`
  [in]: Status of the asynchronous transfer:
  - True: Further callbacks are to be expected.
  - False: This is the last callback.

Example

In the following section, monitored active alarms of the "g_vecAlarmList" vector are acknowledged asynchronously. To this purpose the "AcknowledgeAlarmAsync" function uses a "CAlarmSourceCommandCB" object that implements the "IAlarmSourceCommandCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CAlarmSourceCommandCB" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void AcknowledgeAlarmAsync(IRuntimePtr pRuntime)
{

    ICfUnknownPtr pUnk;
    CFRESULT errCode;
    if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSet"), &pUnk)))
    {
      IAlarmSetPtr pAlarmSet(pUnk);
      CAlarmSourceCommandCB* pAlarmSoureCommand = new CAlarmSourceCommandCB();
      pAlarmSoureCommand->AddRef();
      IAlarmSourceCommandCallback* pCB = pAlarmSoureCommand;
     vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();     // iterate through
list of notified alarms and acknowledge each alarm
      while (it != g_vecAlarmList.end())
      {
        IAlarm* pAlarm = nullptr;  CCfVariant vtAlarmName = it->m_strName;
        errCode = pAlarmSet->Add(vtAlarmName, &pAlarm);
        it++;
      }

      // acknowledged the AlarmSet
      errCode = pAlarmSet->Acknowledge(pCB);
      // wait for acknowledge callback
     if (CF_SUCCEEDED(errCode) && pAlarmSoureCommand->WaitForcompletion(g_nMaxWaitTime) ==
CF_SUCCESS)
      {
        ...
      }
    }
}
```

See also

IAlarmSet (Page 8065)

IAlarmSet

Description

The C++ interface "IAlarmSet" specifies properties and methods for optimized access to several active alarms of the runtime system.

After the initialization of the "IAlarmSet" object, you can execute asynchronous operations with multiple alarms in one call, e. g. acknowledgment or comment. Simultaneous access demonstrates better performance and lower communication load than single access to multiple tags.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"GetCount" method

Return number of alarms of an AlarmSet list.

```
CFRESULT GetCount(uint32_t *value)
```

`value`
[out]: Points to the value for the number of alarms of the AlarmSet list.

"Remove" method

Remove a single alarm or an instance of an alarm from an AlarmSet.

```
CFRESULT Remove(CFSTR name, uint32_t instanceId)
```

- `name`
  [in]: Name of the alarm that is removed to the AlarmSet.
- `instanceId`
  [in]:
  Value = "0": Remove all instances of an active alarm.
  Value > "0": Remove instance with this ID.

"Add" method

Add an active alarm or an instance of the alarm to an AlarmSet.

```
CFRESULT Add(
         CFVARIANT p_varName,
         IAlarm** pAlarm,
         uint32_t instanceId)
```

- `p_varName`
  [in]: Name of the alarm tag that is added to the AlarmSet.
- `pAlarm`
  [out]: Points to the added "IAlarm" object of the AlarmSet.
- `instanceId`
  [in]:
  Value = "0": Add all instances of an active alarm.
  Value > "0": Add instance with this ID.

"Get" method

Reference an alarm or an instance of an alarm from an AlarmSet.

```
CFRESULT Get(
         const CFSTR p_varName,
         IAlarm** pAlarm,
         uint32_t instanceId)
```

- `p_varName`
  [in]: Name of the alarm tag.
- `pAlarm`
  [out]: Points to the "IAlarm" object of the AlarmSet.
- `instanceId`
  [in]:
  Value = "0": Reference all instances of an active alarm.
  Value > "0": Reference instance with this ID.

"Disable" method

Deactivate generation of alarms of the AlarmSet in the alarm source asynchronously.

```
CFRESULT Disable(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Enable" method

Activate generation of alarms of the AlarmSet in the alarm source again asynchronously.

```
CFRESULT Enable(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Shelve" method

Hide alarms of the AlarmSet asynchronously.

```
CFRESULT Shelve(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Unshelve" method

Show hidden alarms of the AlarmSet asynchronously again.

```
CFRESULT Unshelve(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Acknowledge" method

Acknowledge alarms of the AlarmSet asynchronously.

```
CFRESULT Acknowledge(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Reset" method

Acknowledge inactive state of the alarms of the AlarmSet asynchronously. This method is relevant for alarms with the state machine "Alarm with acknowledgment and confirmation". Other state machines do not require a reset.

```
CFRESULT Reset(IAlarmSourceCommandCallback* p_pCallback)
```

`p_pCallback`
[in/out]: Points to an "IAlarmSourceCommandCallback" object that implements the asynchronous operation.

"Clear" method

Remove all alarms from AlarmSets.

```
CFRESULT Clear()
```

Example

Reset of alarms in an alarm list. Reset is the acknowledgment of the inactive status.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;
void ResetAlarmAsync(IRuntimePtr pRuntime)
{
  ICfUnknownPtr pUnk;
  CFRESULT errCode;
  if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmSet"), &pUnk)))
  {
    IAlarmSetPtr pAlarmSet(pUnk);
    CAlarmSourceCommandCB* pAlarmSoureCommond = new CAlarmSourceCommandCB();
    pAlarmSoureCommond->AddRef();
    IAlarmSourceCommandCallback* pCB = pAlarmSoureCommond;
    vector<AlarmAttributes>::iterator it = g_vecAlarmList.begin();     // iterate through
list of notified alarms and reset each alarm
    while (it != g_vecAlarmList.end())
    {
      IAlarm* palarm = nullptr;
      CCfVariant vtAlarmName = it->m_strName;
      errCode = pAlarmSet->Add(vtAlarmName, &palarm);
      it++;
    }
    // Reset the AlarmSet
    errCode = pAlarmSet->Reset(pCB);
    //Check if an alarm could not be reset
    if (CF_SUCCEEDED(errCode) && pAlarmSoureCommond->WaitForcompletion(g_nMaxWaitTime) ==
CF_SUCCESS)
    {
      ...
    }
  }
}
```

See also

IAlarmSourceCommandCallback (Page 8062)

IAlarmSetResult (Page 8069)

IAlarmSetResultEnumerator (Page 8070)

IAlarmSetResult

Description

The C++ interface "IAlarmSetResult" specifies methods for accessing properties of monitored alarms in AlarmSets.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"GetSystemName" method

Return system name of the runtime system of an alarm in the AlarmSet.

```
CFRESULT GetSystemName(CFSTR *value)
```

`value`
[out]: Points to the associated system name.

"GetName" method

Return name of an alarm in the AlarmSet.

```
CFRESULT GetName(CFSTR *value)
```

`value`
[out]: Points to the name of an alarm

"GetInstanceID" method

Return InstanceID of an alarm in the AlarmSet.

```
CFRESULT GetInstanceID(uint32_t *value)
```

`value`
[out]: Points to the InstanceID of an alarm

"GetErrorCode" method

Return error code of an alarm in the AlarmSet.

```
CFRESULT GetErrorCode(uint32_t *value)
```

`value`
[out]: Points to the error code of an alarm.

See also

IAlarmSet (Page 8065)

IAlarmSetResultEnumerator (Page 8070)

IAlarmSubscription (Page 8077)

IAlarmSetResultEnumerator

Description

The "IAlarmSetResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of monitored alarms of the runtime system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IAlarmSetResult **ppItem)
```

`ppItem`
[out]: Points to the current "IAlarmSetResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

See also

IAlarmSet (Page 8065)

IAlarmSetResult (Page 8069)

IAlarmTrigger

Description

The C++ interface "AlarmTrigger" specifies methods for triggering alarms.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"CreateSystemAlarm" method

Creates an alarm of the SystemAlarmWithoutClearEvent class with the state machine alarm without "Inactive with acknowledgment" status.

```
CFRESULT CreateSystemAlarm(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- `alarmText`
  [in]: The alarm text. You have the following options:
  - Transfer a text list of type "ITextList".
    The list entries of the text list can contain the multilingual text or references to other text lists.

    Note

    Only user-defined text lists

    This method processes only user-defined text lists.

  - Transfer static string with monolingual text.
- `area`
  [in]: The area of origin of the alarm
- `p_AlarmParameterValue`<Number>
  [in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEvent:SystemAlarmWithoutClearEvent`
- For monolingual alarm text:
  `@ScriptingSystemEvents.SystemAlarmWithoutClearEventText:SystemAlarmWithoutClearEvent`

"CreateSystemInformation" method

Creates an alarm of the SystemInformation class with the state machine alarm without "Inactive without acknowledgment" status.

```
CFRESULT CreateSystemInformation(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
```

```
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- `alarmText`
  [in]: The alarm text. You have the following options:
  - Transfer a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-defined text lists

    This method processes only user-defined text lists.

  - Transfer static string with monolingual text.
- `area`
  [in]: The area of origin of the alarm
- `p_AlarmParameterValue`<Number>
  [in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts: `@ScriptingSystemEvents.SystemInformation:SystemInformation`
- For monolingual alarm text: `@ScriptingSystemEvents.SystemInformationText:SystemInformation`

"CreateOperatorInputInformation" method

Creates an alarm of the OperatorInputInformation class with the state machine alarm without "Inactive without acknowledgment" status.

```
CFRESULT CreateOperatorInputInformation(
CFVARIANT alarmText,
CFSTR area,
CFVARIANT p_AlarmParameterValue1,
CFVARIANT p_AlarmParameterValue2,
CFVARIANT p_AlarmParameterValue3,
CFVARIANT p_AlarmParameterValue4,
CFVARIANT p_AlarmParameterValue5,
```

```
CFVARIANT p_AlarmParameterValue6,
CFVARIANT p_AlarmParameterValue7)
```

- `alarmText`
  [in]: The alarm text. You have the following options:
  - Transfer a text list of the type "ITextList".
    The list entries of the text list can directly contain multilingual alarm texts or references to other text lists with multilingual alarm texts.

    Note

    Only user-defined text lists

    This method processes only user-defined text lists.

  - Transfer static string with monolingual text.
- `area`
  [in]: The area of origin of the alarm
- `p_AlarmParameterValue`<Number>
  [in]: User-defined comments

The alarm triggers an event with the following event path:

- For multilingual alarm texts:
  `@ScriptingSystemEvents.OperatorInputInformation:OperatorInputInformation`
- For monolingual alarm text:
  `@ScriptingSystemEvents.OperatorInputInformationText:OperatorInputInformation`

Example

The following code examples show how to trigger alarms of the class SystemAlarmWithoutClearEvent. Alarms of the classes SystemInformation and OperatorInputInformation are triggered in the same way.

Copying code

```
void CreateSystemAlarm(IRuntimePtr pRuntime)
{
  ICfUnknownPtr pUnk;
  if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnk)))
  {
    IAlarmTriggerPtr pTrigger(pUnk);
    if (pTrigger != nullptr)
    {
      pTrigger->CreateSystemAlarm(CCfVariant(L"Alarm Text"), CCfString(L"Alarm Area"),
CCfVariant(L"param1"), CCfVariant(L"param2"), CCfVariant(L"param3"),
CCfVariant(L"param4"), CCfVariant(L"param5"), CCfVariant("param6"), CCfVariant("param7"));
    }
  }
}
```

An alarm with multilingual text is created in the following code sample. The alarm text is retrieved from a text list. The alarm text contains a dynamic value (@1%d@), which is transferred via associated value 1.

Copy code

```
void CreateSystemAlarmWithAlarmTextAsTextList(IRuntimePtr pRuntime)
{
  ICfUnknownPtr pUnk;
  if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnk)))
  {
    IAlarmTriggerPtr pTrigger(pUnk);
    if (pTrigger != nullptr)
    {
      if (CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"), &pUnk)))
      {
        ITextListPtr ptextList(pUnk);
        if (nullptr != ptextList)
        {
          // Text list: MyTextList
          // Test list entry index: 101
          // Text: "My input msg. input value = @1%d@"
          ptextList->SetName(CCfString(L"MyTextList"));
          ptextList->SetTextListEntryIndex(101);
          pTrigger->CreateSystemAlarm(CCfVariant(ptextList), CCfString(L"Alarm Area"),
CCfVariant(L"125"),// dynamic value for format specifier @1%d@;
              CCfVariant(), CCfVariant(), CCfVariant(), CCfVariant(), CCfVariant(),
CCfVariant());
        }
      }
    }
  }
}
```

An alarm with multilingual text is created in the following code sample. The alarm text is retrieved from a text list. In turn, the alarm text contains a reference to a second text list and

a dynamic value (@3%s@), which is transferred via associated value 3. (Use case: Nested text list)

Copy code

```
void CreateSystemAlarmWithTextListAsParameterValue(IRuntimePtr pRuntime)
{
  ICfUnknownPtr pUnk;
  if (CF_SUCCEEDED(pRuntime->GetObject(CCfString(L"AlarmTrigger"), &pUnk)))
  {
    IAlarmTriggerPtr pTrigger(pUnk);
    if (pTrigger != nullptr)
    {
      ICfUnknownPtr pUnktextList, pUnktextList1;
      if (CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"), &pUnktextList)) &&
CF_SUCCEEDED(pRuntime->GetObject(CCfString("TextList"), &pUnktextList1)))
      {
        ITextListPtr pTextList_1(pUnktextList);
        if (nullptr != pTextList_1)
        {
          pTextList_1->SetName(CCfString(L"Text_List_1"));
          pTextList_1->SetTextListEntryIndex(1); //Text with reference to Text_List_2:
@1%t#2T@ Val: @3%s@
        }
        ITextListPtr pTextList_2(pUnktextList1);
        if (nullptr != pTextList_2)
        {
          pTextList_2->SetName(CCfString(L"Text_List_2"));
        }
        pTrigger->CreateSystemAlarm(CCfVariant(pTextList_1), CCfString(L"Alarm Area"),
CCfVariant(1), // Index for Text_list_2
            CCfVariant(pTextList_2), // Text_list_2 object
            CCfVariant(L"Hello"), // Dynamic value of @3%s@
            CCfVariant(), CCfVariant(), CCfVariant(), CCfVariant());
      }
    }
  }
}
```

ITextList

Description

The C++ interface "ITextList" is used to transfer multilingual alarm texts for system alarms and operator input alarms. See section IAlarmTrigger (Page 8071), CreateSystemInformation method. An ITextList instance is passed to the alarm text. When the operator input alarm is generated, it is replaced by the configured text from the text list.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Returns the name of the text list.

```
GetName(CFSTR* name)
```

- `name`:
  [out]: The name

"SetName" method

Set the name of the text list.

```
SetName(CFSTR name)
```

- `name`:
  [in]: The name

"GetTextListEntryIndex" method

Return the index of the list entry.

```
GetTextListEntryIndex)(OUT uint32_t* pIndex)
```

- `pIndex`
  [out]: The index

"SetTextListEntryIndex" method

Set the index of the list entry.

```
SetTextListEntryIndex)(IN uint32_t pIndex)
```

- `pIndex`
  [in]: The index

IAlarmSubscription

Description

The C++ interface "IAlarmSubscription" specifies methods for monitoring tags of the Runtime system. The subscribed tags are monitored for a change of the alarm state.

Members

The following methods are specified in the interface:

"Start" method

Start monitoring of active alarms.

```
CFRESULT Start(IAlarmCallback* callbackPtr)
```

`callbackPtr`
[in/out]: Points to an "IAlarmCallback" object that implements the asynchronous monitoring.

"Stop" method

Stop monitoring of active alarms.

Note

Start and Stop in Windows Forms applications

Do not call the "Stop" method for a Windows Forms application in the same thread where you called "Start".

```
CFRESULT Stop()
```

"SetSystemNames" method

Set system names of runtime systems for the monitoring of active alarms.

```
CFRESULT SetSystemNames(CFVARIANT systemIDs)
```

`systemIDs`
[in]: System names of Runtime systems

"GetSystemNames" method

Read out system names of runtime systems for the monitoring of active alarms.

```
CFRESULT GetSystemNames(CFVARIANT* systemIDs)
```

`systemIDs`
[out]: Points to system names of Runtime systems.

"SetLanguage" method

Set country identification of the language for monitored alarms.

```
CFRESULT SetLanguage(uint32_t language)
```

`language`
[in]: Country identification of the language

"GetLanguage" method

Read out country identification of the language for monitored alarms.

```
CFRESULT GetLanguage(uint32_t* language)
```

`language`
[out]: Points to country identification of the language.

"GetFilter" method

Supplies the string by which the result set is filtered.

```
CFRESULT GetFilter(CFSTR* filter)
```

- `filter`
  [out]: SQL-type string for filtering the result set of active alarms.

"SetFilter" method

Sets the string for filtering the result set of active alarms.

```
CFRESULT SetFilter(IN CFSTR filter)
```

- `filter`
  [in]: SQL-type string for filtering the result set of active alarms.
  All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 7828).

Example

In the following section, monitored active alarms are written asynchronously into the "g_vecAlarmList" map at a change. To this purpose the "SubscribeAlarm" function uses a "CAlarmValue" object that implements the "IAlarmCallback" interface and that uses the "COdkTagSourceCBBase" class. The service life of the "CAlarmValue" object is determined via reference counting.

Copy code

```
const uint32_t g_nMaxWaitTime = 6000;
vector<AlarmAttributes> g_vecAlarmList;

void SubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    pRuntime->GetObject(CCfString(L"AlarmSubscription"), &pUnk);

    IAlarmSubscriptionPtr pAlarm(pUnk);
    CAlarmValue* pAlarmValue = new CAlarmValue();
    pAlarmValue->AddRef();
    IAlarmCallback *pCB = pAlarmValue;

    CCfSafeArrayBound bounds(1UL, 0);

    CCfSafeArray daAttribute;
    CCfSafeArray daSystemID(CF_VT_SREF, 1, &bounds);
    CCfVariant daDataSource = 0;
    CCfVariant vSystemIDs = 0;

    CCfSmartString daFilter = L"";

    CCfSREF id(L"SYSTEM1");
    int32_t index = 0;
    daSystemID.PutElement(&index, &id);

    CCfVariant daLanguage = 1033;

    daSystemID.Detach(&vSystemIDs);

    CCfSmartString strFilter = L"";
    // Start Subscription
    pAlarm->SetFilter(strFilter.AllocCFSTR());
    pAlarm->SetLanguage(1033);
    pAlarm->SetSystemNames(vSystemIDs);

    pAlarm->Start(pCB);

    // Wait for alarm notifications
    if (pAlarmValue->WaitForcompletion(g_nMaxWaitTime) == CF_SUCCESS)
    {
       pAlarm->Stop();
       // Get current alarms from callback
       pAlarmValue->GetAlarmAttributes(g_vecAlarmList);
    }
}
```

See also

IAlarmResult (Page 8046)

IAlarmCallback (Page 8059)

IAlarmSetResult (Page 8069)

ILoggedAlarmResult

Description

The C++ interface "ILoggedAlarmResult" specifies methods for accessing properties of logged alarms of a logging system.

An "ILoggedAlarmResult" object is a pure data object that maps all properties of a logged alarm.

Members

The following methods are specified in the interface:

"GetInstanceID" method

Return InstanceID of a logged alarm.

```
CFRESULT GetInstanceID(uint32_t *value)
```

`value`
[out]: Points to the InstanceID of the logged alarm.

"GetName" method

Return the name of the logged alarm.

```
CFRESULT GetName(CFSTR *value)
```

`value`
[out]: Points to the name of the logged alarm.

"GetAlarmClassName" method

Return the name of the alarm class.

```
CFRESULT GetAlarmClassName(CFSTR *value)
```

`value`
[out]: Points to the symbol of the alarm class of the logged alarm.

"GetAlarmClassSymbol" method

Return the symbol of the alarm class.

```
CFRESULT GetAlarmClassSymbol(CFSTR *value)
```

`value`
[out]: Points to the name of the alarm class of the logged alarm.

"GetState" method

Return alarm state of the logged alarm.

```
CFRESULT GetState(int32_t* *value)
```

`value`
[out]: Points to the alarm state.

"GetStateText" method

Return alarm state of the logged alarm as text, for example, "active" or "inactive".

```
CFRESULT GetStateText(CFSTR *value)
```

`value`
[out]: Points to the alarm state as text.

"GetEventText" method

Return text that describes the alarm event of the logged alarm.

```
CFRESULT GetEventText(CFSTR *value)
```

`value`
[out]: Points to the text that describes the alarm event.

"GetAlarmText1GetAlarmText9" method

Return additional texts 1-9 of the logged alarm.

```
CFRESULT GetAlarmText1(CFSTR *value)
```

...

```
CFRESULT GetAlarmText9(CFSTR *value)
```

`value`
[out]: Points to the additional text of the logged alarm.

"GetTextColor" method

Return text color of the alarm state of the logged alarm.

```
CFRESULT GetTextColor(uint32_t *value)
```

`value`
[out]: Points to the text color of the alarm state.

"GetBackColor" method

Return background color of the alarm state of the logged alarm.

```
CFRESULT GetBackColor(uint32_t *value)
```

`value`
[out]: Points to the background color of the alarm state.

"GetFlashing" method

Return flashing background color of the alarm state of the logged alarm.

```
CFRESULT GetFlashing(CFBOOL *value)
```

`value`
[out]: Points to the flashing background color of the alarm state.

"GetModificationTime" method

Return time of the last modification to the alarm state of the logged alarm.

```
CFRESULT GetModificationTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the last change of the alarm state.

"GetChangeReason" method

Return trigger event for modification of the alarm state of the logged alarm.

```
CFRESULT GetChangeReason(uint16_t *value)
```

`value`
[out]: Points to the trigger event for the change of the alarm state.

"GetRaiseTime" method

Return trigger time of the logged alarm.

```
CFRESULT GetRaiseTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time the logged alarm was triggered.

"GetAcknowledgementTime" method

Return time of alarm acknowledgment of the logged alarm.

```
CFRESULT GetAcknowledgementTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of acknowledgment of the logged alarm.

"GetClearTime" method

Return time of reset of the logged alarm.

```
CFRESULT GetClearTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time when the logged alarm is reset.

"GetResetTime" method

Return time of reset of the logged alarm.

```
CFRESULT GetResetTime(CFDATETIME64 *value)
```

`value`
[out]: Points to the time of the reset of the logged alarm.

"GetSuppressionState" method

Return status of visibility of the logged alarm.

```
CFRESULT GetSuppressionState(uint8_t *value)
```

`value`
[out]: Points to the status of the visibility of the logged alarm.

"GetPriority" method

Return relevance for display and sorting of the logged alarm.

```
CFRESULT GetPriority(uint8_t *value)
```

`value`
[out]: Points to the relevance of the logged alarm.

"GetOrigin" method

Return origin for display and sorting of the logged alarm.

```
CFRESULT GetOrigin(CFSTR *value)
```

`value`
[out]: Points to the origin of the logged alarm.

"GetArea" method

Return origin area for display and sorting of the logged alarm.

```
CFRESULT GetArea(CFSTR *value)
```

`value`
[out]: Points to the origin area of the logged alarm.

"GetValue" method

Return process value of the logged alarm.

```
CFRESULT GetValue(CFVARIANT *value)
```

`value`
[out]: Points to the process value of the logged alarm.

"GetValueQuality" method

Return quality of the process value of the logged alarm.

```
CFRESULT GetValueQuality(uint16_t *value)
```

`value`
[out]: Points to the quality of the process value of the logged alarm.

"GetValueLimit" method

Return process value limit of the logged alarm.

```
CFRESULT GetValueLimit(CFVARIANT *value)
```

`value`
[out]: Points to the limit of the process value of the logged alarm.

"GetUserName" method

Return the user name of the logged operator input alarm.

```
CFRESULT GetUserName(CFSTR *value)
```

`value`
[out]: Points to the user name of the logged operator input alarm.

"GetLoopInAlarm" method

Return function that navigates from the alarm control to its origin.

```
CFRESULT GetLoopInAlarm(CFSTR *value)
```

`value`
[out]: Points to the function name that navigates to the origin of the logged alarm.

"GetAlarmParameterValues" method

Return parameter values of the logged alarm.

```
CFRESULT GetAlarmParameterValues(CFVARIANT *value)
```

`value`
[out]: Points to the parameter values of the logged alarm.

"GetInvalidFlags" method

Return identification of the logged alarm with invalid data.

```
CFRESULT GetInvalidFlags(uint8_t *value)
```

`value`
[out]: Points to the invalid data of the logged alarm.

"GetConnection" method

Return connection via which the logged alarm was triggered.

```
CFRESULT GetConnection CFSTR *value)
```

`value`
[out]: Points to the connection of the logged alarm.

"GetSystemSeverity" method

Return severity of the system error.

```
CFRESULT GetSystemSeverity(uint8_t *value)
```

`value`
[out]: Points to the severity of the system error.

"GetUserResponse" method

Return expected or required user response to the logged alarm.

```
CFRESULT GetUserResponse(uint8_t *value)
```

`value`
[out]: Points to the expected or required user response to the logged alarm.

"GetDeadBand" method

Return range of the triggering tag in which no alarms are generated.

```
CFRESULT GetDeadBand(CFVARIANT *value)
```

`value`
[out]: Points to the deadband of the logged alarm.

"GetHostName" method

Return the name of the host that triggered the alarm.

```
CFRESULT GetHostName(CFSTR *value)
```

`value`
[out]: Points to host name.

"GetInfoText" method

Return text for the alarm that contains the associated work instruction.

```
CFRESULT GetInfoText(CFSTR *value)
```

`value`
[out]: Points to the text of the operator instruction.

"GetStateMachine" method

Return StateMachine model of the alarm. The StateMachine represents the behavior of alarms through the arrangement of alarm states and alarm events, e.g. "RaiseClear", "RaiseRequiresAcknowledgment" or "RaiseClearOptionalAcknowledgment".

```
CFRESULT GetStateMachine(uint8_t *value)
```

`value`
[out]: Shows the model of the StateMachine of the logged alarm.

"GetSingleAcknowledgement" method

Returns whether an alarm may be acknowledged only individually or may be acknowledged as a group or multiple selection.

```
CFRESULT GetSingleAcknowledgement(CFBOOL *value)
```

`value`
[out]: Points to the acknowledgement specification.

"GetLoggedAlarmStateObjectID" method

Return ID of the alarm state for referencing within the logging system.

```
CFRESULT GetLoggedAlarmStateObjectID(CFSTR *value)
```

`value`
[out]: Points to the ID of the alarm state of the logged alarm.

"GetID" method

Return the user-defined ID of the alarm that is also used in the display.

```
CFRESULT GetID(uint32_t *value)
```

`value`
[out]: Points to the ID of the logged alarm.

"GetSourceType" method

Return source from which the alarm was generated, for example, tag-based, controller-based or system-based alarm.

```
CFRESULT GetSourceType(uint16_t *value)
```

`value`
[out]: Points to the type of source of the logged alarm.

Method "GetDuration"

Returns the time interval in nanoseconds between triggering of the logged alarm and its previous status change.

```
CFRESULT GetDuration(CFTIMESPAN64 *value)
```

`value`
[out]: The time interval in nanoseconds

See also

ILoggedAlarmResultEnumerator (Page 8087)

IAlarmLogging (Page 8088)

IAlarmLoggingSubscription (Page 8092)

ILoggedAlarmResultEnumerator

Description

The "ILoggedAlarmResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of logged alarms of a logging system. Through the enumeration, you access individual alarms from the set of logged alarm of a logging system.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(ILoggedAlarmResult **ppItem)
```

`ppItem`
[out]: Points to the current "ILoggedAlarmResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *value)
```

`value`
[out]: Points to the value for the number of elements of the list.

See also

ILoggedAlarmResult (Page 8081)

IAlarmLogging (Page 8088)

IAlarmLogging

Description

The C++ interface "IAlarmLogging" specifies methods for reading out logged alarms of a logging system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"Read" method

Read out logged alarms of a time period synchronously from logging system.

```
CFRESULT Read(
         CFDATETIME64 begin,
         CFDATETIME64 end,
         CFSTR filter,
         uint32_t language,
         CFVARIANT systemIDs,
         ILoggedAlarmResultEnumerator **ppEnumerator)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `filter`
  [in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.

- `language`
  [in]: Country identification of the language of the logged alarm text
- `systemIDs`
  [in]: System names of the runtime systems of the logged alarms. Default: local system
- `ppEnumerator`
  [out]: Points to the logged alarms as an "ILoggedAlarmResultEnumerator" object.

"ReadAsync" method

Read out logged alarms of a time period asynchronously from logging system.

```
CFRESULT ReadAsync(
         CFDATETIME64 begin,
         CFDATETIME64 end,
         CFSTR filter,
         uint32_t language,
         CFVARIANT systemIDs,
         IAlarmLoggingCallback *pLoggedAlarmCb)
```

- `begin`
  [in]: Start date of the time period
- `end`
  [in]: End date of the time period
- `filter`
  [in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `language`
  [in]: Country identification of the language of the logged alarm text
- `systemIDs`
  [in]: System names of the runtime systems of the logged alarms. Default: local system
- `pLoggedAlarmCb`
  [in]: Points to the "IAlarmLoggingCallback" object that implements the callback interface.

Example

Read historical alarms synchronously from the logging system:

Copy code

```
void LoggingReadAlarmSync(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    pRuntime->GetObject(CCfString(L"LoggedAlarm"), &pUnk);

    CCfVariant vSystemIDs = 0;
    CFSAFEARRAYBOUND* bounds = nullptr;
    CCfSafeArray daSystemID(CF_VT_SREF, 1, bounds);
    CCfSREF id(L"SYSTEM1");
    int32_t index = 0;
    daSystemID.PutElement(&index, &id);
    daSystemID.Detach(&vSystemIDs);

    IAlarmLoggingPtr pAlarm(pUnk);

    CCfDateTime64 begin, end;
    begin = CCfDateTime64::Now(true);
    end = begin;
    begin.SubtractTimeSpan(Get1Minute() * 3);

    ILoggedAlarmResultEnumeratorPtr pItems;
    // Read value of tag
    pAlarm->Read(begin, end, CCfString(""), 1033, vSystemIDs, &pItems);

    std::wcout << "Read finished " << std::endl;

    // Iterate over the process value objects and print them
    while (CF_SUCCEEDED(pItems->MoveNext()))
    {
        ILoggedAlarmResultPtr pValue;
        pItems->Current(&pValue); // get current process value
        CCfString timeStamp;
        CFDATETIME64 cfTimeStamp;
        pValue->GetModificationTime(&cfTimeStamp);
        CCfDateTime64 time(cfTimeStamp);
        timeStamp = time.GetDateTimeString(false);
        CCfString strName;
        pValue->GetName(&strName);
        CCfVariant varValue;
        pValue->GetValue(&varValue);
        std::wcout << strName.ToUTF8().c_str() << L" " << timeStamp.ToUTF8().c_str() << L"
Value: " << (double)(varValue) << std::endl;
    }
}
```

See also

ILoggedAlarmResult (Page 8081)

ILoggedAlarmResultEnumerator (Page 8087)

IAlarmLoggingCallback

Description

The C++ interface "IAlarmLoggingCallback" defines methods for implementing asynchronous operations for monitoring active alarms. The methods are used by the "IAlarmLogging" and "IAlarmLoggingSubscription" interfaces.

All the methods return CF_SUCCESS after execution.

Members

The following methods are specified in the "IAlarmCallback" interface:

"OnReadComplete" method

Callback method is called on completion of asynchronous read operations in logging systems.

The "OnReadComplete" callback method is called when the "IAlarmLogging.ReadAsync" method is used.

```
CFRESULT OnReadComplete(ILoggedAlarmResultEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegAlarmResultEnumerator" object that contains the enumeration of the logged alarms.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logged alarms.

"OnDataChanged" method

Callback method is called upon a change of a monitored alarm in logging systems.

The "OnDataChanged" callback method is called when the "IAlarmLoggingSubscription.Start" method is used.

```
CFRESULT OnDataChanged(ILoggedAlarmResultEnumerator *pEnumerator,
         uint32_t errorCode,
         int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "ILoggegAlarmResultEnumerator" object that contains the enumeration of the logged alarms.
- `errorCode`
  [out]: Error code for the asynchronous operation
- `contextId`
  [out]: ContextID as additional identification feature of the logged alarms.

See also

IAlarmLogging (Page 8088)

IAlarmLoggingSubscription (Page 8092)

IAlarmLoggingSubscription

Description

The C++ interface "IAlarmLoggingSubscription" specifies methods for monitoring logged alarms of a logging system.

All the methods return CF_SUCCESS following successful execution. In the case of an error, the methods return the corresponding error code.

Members

The following methods are specified in the interface:

"SetFilter" method

Set SQL-type string for filtering the result set of the logged alarms.

```
CFRESULT SetFilter(CFSTR filter)
```

`filter`
[in]: Filter string for logged alarms

"SetLanguage" method

Set country identifier of the language for monitoring of logged alarms.

```
CFRESULT SetLanguage(uint32_t language)
```

`language`
[in]: Country identification of the language

"SetSystemName" method

Set system names of runtime systems for the monitoring of logged alarms.

```
CFRESULT SetSystemName(CFVARIANT systemIDs)
```

`systemIDs`
[in]: System name of Runtime systems

"Start" method

Start monitoring of logged alarms.

```
CFRESULT Start(IAlarmLoggingCallback* pLoggedAlarmCb)
```

- `filter`
  [in]: Filter for limiting the read operation with properties of the "ILoggedAlarmResult" object.
- `pLoggedAlramCb`
  [in/out]: Points to an "IAlarmLoggingCallback" object that implements asynchronous monitoring.

"Stop" method

Stop monitoring of all logged alarms.

```
CFRESULT Stop()
```

Example

Monitoring historical alarms. The values are returned by the "IAlarmLoggingCallback" object:

Copy code

```
void LoggingSubscribeAlarm(IRuntimePtr pRuntime)
{
    ICfUnknownPtr pUnk;
    pRuntime->GetObject(CCfString(L"LoggedAlarmSubscribtion"), &pUnk);

    CCfVariant vSystemIDs = 0;
    CFSAFEARRAYBOUND* bounds = nullptr;
    CCfSafeArray daSystemID(CF_VT_SREF, 1, bounds);
    CCfSREF id(L"SYSTEM1");
    int32_t index = 0;
    daSystemID.PutElement(&index, &id);
    daSystemID.Detach(&vSystemIDs);

    IAlarmLoggingSubscriptionPtr pAlarm(pUnk);
    pAlarm->SetSystemName(vSystemIDs);
    pAlarm->SetLanguage(1033);

    COdkAlarmLoggingCB* pAlarmCB = new COdkAlarmLoggingCB();

    pAlarmCB->AddRef();

    // subscribe tags
    pAlarm->Start(pAlarmCB);
}
```

See also

IAlarmLogging (Page 8088)

IAlarmLoggingCallback (Page 8091)

ILoggedAlarmResult (Page 8081)

20.3.9.5 Interfaces for connections

IConnectionResult

Description

The C++ interface "IConnectionResult" provides methods for access to the details of connections.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

`pName`
[out]: Points to the name of the connection.

"GetConnectionState" method

Return status of the connection.

```
CFRESULT GetConnectionState(CFENUM *pConnectionState)
```

`pConnectionState`
[out]: Points to the enumeration, which can contain the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"GetEstablishmentMode" method

Return mode in which the connection is established.

```
CFRESULT GetEstablishmentMode(CFENUM *pEstablishmentMode)
```

`pEstablishmentMode`
[out]: Points to the enumeration, which can contain the following values:

- `None` (0)
- `AutomaticActive` (1)
- `AutomaticPassive` (2)
- `OnDemandActive` (3)
- `OnDemandPassive` (4)

"GetTimeSynchronizationMode" method

Mode of time synchronization between HMI system and AS.

```
CFRESULT GetTimeSynchronizationMode(CFENUM
*pTimeSynchronizationMode)
```

`pTimeSynchronizationMode`
[out]: Points to the enumeration, which can contain the following values:

- `None` (0)
- `Subordinate` (1)
- `Lead` (2)

"GetDisabledAtStartup" method

Indicates whether the connection is disabled at the start of Runtime.

```
CFRESULT GetDisabledAtStartup(CFBOOL *pDisabledAtStartup)
```

`pDisabledAtStartup`
[out]: Points to a Boolean value.

"GetEnabled" method

Indicates whether the connection is active.

```
CFRESULT GetEnabled(CFBOOL *pEnabled)
```

`pbEnabled`
[out]: Points to a Boolean value.

"GetConnectionType" method

Return protocol of a communication driver, e.g. "S7 Classic".

```
CFRESULT GetConnectionType(CFSTR *pConnectionType)
```

`pConnectionType`
[out]: Points to the name of the protocol.

"GetError" method

Return error code of the connection.

```
CFRESULT GetError(uint32_t *pError)
```

`pError`
[out]: Points to the error code.

Example

Output connection details:

Copy code

```
void DisplayConnectionInfo(IConnectionResultPtr pConnectionresult,IRuntimePtr pRuntime)
{
   CCfString strName;
   pConnectionresult->GetName(&strName);
   std::cout << "ConnectionName:" << strName.ToUTF8() << std::endl;
   CCfString strConnectiontype;
   pConnectionresult->GetConnectionType(&strConnectiontype);
   std::cout << "ConnectionType:" << strConnectiontype.ToUTF8() << std::endl;
   CFENUM enConnectionState;
   pConnectionresult->GetConnectionState(&enConnectionState);
   HmiConnectionState enumconnectionState =
static_cast<HmiConnectionState>(enConnectionState);
   ConnectionState(enumconnectionState);
   CFENUM enEstablishmentMode;
   pConnectionresult->GetEstablishmentMode(&enEstablishmentMode);
   HmiConnectionEstablishmentMode enumEstablishmentMode =
static_cast<HmiConnectionEstablishmentMode>(enEstablishmentMode);
   Establishmentmode(enumEstablishmentMode);
   CFENUM enTimeSynchronizationMode;
   pConnectionresult->GetTimeSynchronizationMode(&enTimeSynchronizationMode);
   HmiTimeSynchronizationMode enumTimeSynchronizationMode =
static_cast<HmiTimeSynchronizationMode>(enTimeSynchronizationMode);
   TimeSynchronizationmode(enumTimeSynchronizationMode);
   CFBOOL bDisableatStartup;
   pConnectionresult->GetDisabledAtStartup(&bDisableatStartup);
   std::cout << "DisableStartup:" << (int)bDisableatStartup << std::endl;
   CFBOOL bEnabled;
   pConnectionresult->GetEnabled(&bEnabled);
   std::cout << "Enabled:" << (int)bEnabled << std::endl;
}
```

See also

IConnection (Page 8101)

IConnectionResultEnumerator (Page 8096)

IConnectionResultEnumerator

Description

The "IConnectionResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of connection details of the Runtime system. The enumeration is used, for example, when reading out connections of a connection set.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IConnectionResult **ppItem)
```

ppItem
[out]: Points to the current "IConnectionResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *pCount)
```

pCount
[out]: Points to the value for the number of elements of the list.

See also

IConnectionResult (Page 8094)

IConnectionSet (Page 8104)

IConnectionStatusResult

Description

The C++ interface "IConnectionStatusResult" provides methods for access to the status of connections.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

`pName`
[out]: Points to the name of the connection.

"GetConnectionStatus" method

Return status of the connection.

```
CFRESULT GetConnectionStatus(CFENUM *pConnectionStatus)
```

`pConnectionStatus`
[out]: Points to the enumeration, which can contain the following values:

- `Disabled` (0)
- `Connecting` (1)
- `Connected` (2)
- `Disconnecting` (3)
- `Disconnected` (4)
- `Reconnecting` (5)

"GetError" method

Return error code of the connection.

```
CFRESULT GetError(uint32_t *pError)
```

`pError`
[out]: Points to the error code.

Example

Output status of a certain connection:

Copy code

```
void ConnectionSet_GetConnectionState(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);

   IConnectionSetPtr pConnectionsetPtr(pUnk);
   CCfSmartString strName(L"HMI-Connection");
   CCfString strName1(L"HMI-ConnectionS7Plus");
   CCfArrayVariant vtArrayVaiarnt;
   vtArrayVaiarnt.Append(strName);
   vtArrayVaiarnt.Append(strName1);
   ICfArrayVariantPtr connectionNames;
   vtArrayVaiarnt.DetachEnumerator(&connectionNames);
   pConnectionsetPtr->Add(connectionNames);

   IConnectionStatusResultEnumeratorPtr pConnectionStatusResultEnum;
   IConnectionStatusResultPtr pConnectionStatusresult;
   pConnectionsetPtr->GetConnectionState(&pConnectionStatusResultEnum);

   uint32_t nCount;
   pConnectionStatusResultEnum->Count(&nCount);
   for (int32_t i = 0; i < (int32_t)nCount; i++)
   {
      pConnectionStatusResultEnum->MoveNext();
      pConnectionStatusResultEnum->Current(&pConnectionStatusresult);

      CCfString strName;
      pConnectionStatusresult->GetName(&strName);
      std::cout << "ConnectionName:" << strName.ToUTF8() << std::endl;
      CFENUM enConnectionState;
      pConnectionStatusresult->GetConnectionStatus(&enConnectionState);
      HmiConnectionState enumconnectionState =
static_cast<HmiConnectionState>(enConnectionState);
      ConnectionState(enumconnectionState);
   }
}
```

See also

IConnection (Page 8101)

IConnectionStatusResultEnumerator (Page 8100)

IConnectionStatusResultEnumerator

Description

The "IConnectionStatusResultEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of connection status of the Runtime system. The enumeration is used, for example, when reading out connections of a connection set.

All the methods return CF_SUCCESS following successful execution.

Members

The following methods are specified in the interface:

"Current" method

Output the current element of the enumeration of a list.

```
CFRESULT Current(IConnectionStatusResult **ppItem)
```

`ppItem`
[out]: Points to the current "IConnectionStatusResult" object as an element of the list.

"MoveNext" method

Go to the next element of the enumeration of a list.

```
CFRESULT MoveNext()
```

"Reset" method

Reset the current position in the enumeration of a list.

```
CFRESULT Reset()
```

The "MoveNext" method moves afterwards to the first element of the list.

"Count" method

Output the size of the enumeration or the number of elements of a list.

```
CFRESULT Count(uint32_t *pCount)
```

`pCount`
[out]: Points to the value for the number of elements of the list.

See also

IConnectionStatusResult (Page 8097)

IConnectionSet (Page 8104)

IConnection

Description

The C++ interface "IConnection" provides properties and methods for access to a connection.

Members

The following methods are specified in the interface:

"GetName" method

Return name of the connection.

```
CFRESULT GetName(CFSTR *pName)
```

`pName`
[out]: Points to the name of the connection.

"SetName" method

Change name of the connection.

```
CFRESULT SetName(CFSTR name)
```

`name`
[in]: Name of the connection

"Read" method

Read connection details synchronously from the Runtime system.

```
CFRESULT Read(IConnectionResult **ppConnectionResult)
```

`ppConnectionResult`
[out]: Points to an object of type "IConnectionResult" that contains the connection details.

"GetConnectionState" method

Return connection status of a connection.

```
CFRESULT GetConnectionState(IConnectionStatusResult
**ppConnectionStatusResult)
```

`ppConnectionStatusResult`
[out]: Points to an object of type "IConnectionStatusResult" that contains the status of connections.

"SetConnectionMode" method

Change connection status of a connection.

```
CFRESULT SetConnectionMode(CFENUM connectionmode)
```

`connectionmode`
[in]: Enumeration which contains the mode of connections:

- `Disabled` (0)
- `Enabled` (1)

Examples

Change status of a connection:

Copy code

```
void Connection_SetConnectionState(IRuntimePtr pRuntime, ConnectionMode connectionMode)
{
	ICfUnknownPtr pUnk;
	pRuntime->GetObject(CCfString(L"Connection"), &pUnk);

	IConnectionPtr pConnection(pUnk);
	CCfSmartString daName = L"HMI-Connection";
	pConnection->SetName(daName.AllocCFSTR());
	pConnection->SetConnectionMode(int32_t(connectionMode));
}
```

See also

IConnectionResult (Page 8094)

IConnectionStatusResult (Page 8097)

IConnectionSet (Page 8104)

IConnectionReadNotification

Description

The C++ interface "IConnectionReadNotification" defines a callback method for implementation of operations following read operations of connections.

Members

The following method is specified in the interface:

"OnReadComplete" method

Callback method is called following read operations of connections.

The "OnReadComplete" callback method is called when the `IConnectionSet.ReadAsync` method is used.

```
CFRESULT OnReadComplete(IConnectionResultEnumerator *pEnumerator,
uint32_t errorCode, int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "IConnectionResultEnumerator" object that contains the enumeration of the connection details.
- `errorCode`
  [out]: Error code for the asynchronous operation.
- `contextId`
  [out]: ContextID as additional identification feature of the connections.

Example

Read out connection asynchronously:

Copy code

```
void ConnectionSet_ReadAsync(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
   IConnectionSetPtr pConnectionset(pUnk);
   IConnectionResultPtr pConnectionResult;
   CCfString strName(L"HMI-Connection");
   CCfString strName1(L"HMI-ConnectionS7Plus");
   CCfArrayVariant vtArrayVaiarnt;
   vtArrayVaiarnt.Append(strName);
   vtArrayVaiarnt.Append(strName1);
   ICfArrayVariantPtr connectionNames;
   vtArrayVaiarnt.DetachEnumerator(&connectionNames);
   pConnectionset->Add(connectionNames);
  CCfRefPtr<CConnectionReadNotification>pRead = new CConnectionReadNotification(pRuntime);
   IConnectionReadNotification *pNotification = pRead;
   pConnectionset->ReadAsync(pNotification);
}
```

See also

IConnectionSet (Page 8104)

IConnectionStateChangeNotification

Description

The C++ interface "IConnectionStateChangeNotification" defines a callback method for implementing asynchronous change monitoring of connections.

Members

The following method is specified in the interface:

"OnDataChanged" method

Callback method is called after changes of a monitored connection.

The "OnDataChanged" callback method is called when the `IConnectionSet.Subscribe` method is used.

```
CFRESULT OnDataChanged(IConnectionStatusResultEnumerator*
pEnumerator, uint32_t errorCode, int32_t contextId)
```

- `pEnumerator`
  [out]: Points to an "IConnectionStatusResultEnumerator" object that contains the enumeration of the connection status.
- `errorCode`
  [out]: Error code for the asynchronous operation.
- `contextId`
  [out]: ContextID as additional identification feature of the connections.

Example

Monitor connection status:

Copy code

```
void ConnnectionSet_Subscribe(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
   IConnectionSetPtr pConnectionsetPtr(pUnk);
   CCfString strName(L"RUNTIME_1::Connection3");
   CCfString strName1(L"HMI-ConnectionS7Plus");
   CCfArrayVariant vtArrayVaiarnt;
   vtArrayVaiarnt.Append(strName);
   vtArrayVaiarnt.Append(strName1);
   ICfArrayVariantPtr connectionNames;
   vtArrayVaiarnt.DetachEnumerator(&connectionNames);
   pConnectionsetPtr->Add(connectionNames);
   CConnectionSubscriptionNotification *pSubscribe = new
CConnectionSubscriptionNotification();
   IConnectionStateChangeNotification *pNotification = pSubscribe;
   pConnectionsetPtr->Subscribe(pNotification);
}
```

See also

IConnectionSet (Page 8104)

IConnectionSet

Description

The C++ interface "IConnectionSet" specifies properties and methods for optimized access to several connections of the Runtime system.

After initialization of the "IConnectionSet" object, you have read/write access to multiple connections in one call. Simultaneous access takes place with better performance and lower communication load than single access to multiple connections.

Members

The following methods are specified in the interface:

"SetContextId" method

Change ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

```
CFRESULT SetContextId(uint32_t id)
```

`id`
[in]: ContextID of the connection

"GetContextId" method

Return ID as additional identification feature of a connection. The ContextId can, for example, be used to recognize identically named connections.

```
CFRESULT GetContextId(uint32_t *pId)
```

`pId`
[out]: Points to the ContextID of the connection.

"Add" method

Add connections to a connection set.

```
CFRESULT Add(ICfArrayVariant *connectionNames)
```

`connectionNames`
[in]: Points to an array that contains the names of connections.

"Remove" method

Remove individual connection from connection set.

```
CFRESULT Remove(CFSTR connectionName)
```

`connectionName`
[in]: Name of the connection.

"Clear" method

Remove all connections from connection set.

```
CFRESULT Clear()
```

"GetCount" method

Return number of connections of a connection set list.

```
CFRESULT GetCount(int32_t *pCount)
```

`pCount`
[out]: Points to the number of connections in the connection set.

"Read" method

Read connection details of all connections of the connection set synchronously from the Runtime system.

```
CFRESULT Read(IConnectionResultEnumerator
**ppConnectionResultEnumerator)
```

`ppConnectionResultEnumerator`
[out]: Points to the enumeration of the details of the individual connections.

"ReadAsync" method

Read connection details of all connections of the connection set asynchronously from the Runtime system.

```
CFRESULT ReadAsync(IConnectionReadNotification *pReadReply)
```

`pReadReply`
[in]: Points to the "IConnectionReadNotification" callback interface for read operations and returns the "IConnectionResultEnumerator" object.

"GetConnectionState" method

Read connection status synchronously from the Runtime system.

```
CFRESULT GetConnectionState(IConnectionStatusResultEnumerator
**ppConnectionStatusResultEnumerator)
```

`ppConnectionStatusResultEnumerator`
[out]: Points to an object of type "IConnectionStatusResultEnumerator" that contains the enumeration of the connection status.

"Subscribe" method

Subscribe all connections of a connection set asynchronously for change monitoring.

```
CFRESULT Subscribe(IConnectionStateChangeNotification
*ppNotificationCB)
```

`ppNotificationCB`
[in]: Points to the "IConnectionStateChangeNotification" callback interface for monitoring and returns the "IConnectionStatusResultEnumerator" object following a change.

"CancelSubscribe" method

Cancel change monitoring of all connections of a connection set.

```
CFRESULT CancelSubscribe()
```

Examples

Monitor connection:

Copy code

```
void ConnnectionSet_Subscribe(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
   IConnectionSetPtr pConnectionsetPtr(pUnk);
   CCfString strName(L"RUNTIME_1::Connection3");
   CCfString strName1(L"HMI-ConnectionS7Plus");
   CCfArrayVariant vtArrayVaiarnt;
   vtArrayVaiarnt.Append(strName);
   vtArrayVaiarnt.Append(strName1);
   ICfArrayVariantPtr connectionNames;
   vtArrayVaiarnt.DetachEnumerator(&connectionNames);
   pConnectionsetPtr->Add(connectionNames);
   CConnectionSubscriptionNotification *pSubscribe = new
CConnectionSubscriptionNotification();
   IConnectionStateChangeNotification *pNotification = pSubscribe;
   pConnectionsetPtr->Subscribe(pNotification);
}
```

Read out connection asynchronously:

Copy code

```
void ConnectionSet_ReadAsync(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
   IConnectionSetPtr pConnectionset(pUnk);
   IConnectionResultPtr pConnectionResult;
   CCfString strName(L"HMI-Connection");
   CCfString strName1(L"HMI-ConnectionS7Plus");
   CCfArrayVariant vtArrayVaiarnt;
   vtArrayVaiarnt.Append(strName);
   vtArrayVaiarnt.Append(strName1);
   ICfArrayVariantPtr connectionNames;
   vtArrayVaiarnt.DetachEnumerator(&connectionNames);
   pConnectionset->Add(connectionNames);
  CCfRefPtr<CConnectionReadNotification>pRead = new CConnectionReadNotification(pRuntime);
   IConnectionReadNotification *pNotification = pRead;
   pConnectionset->ReadAsync(pNotification);
}
```

Add/remove connection for connection set

Copy code

```
void Connection_AddRemove(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ConnectionSet"), &pUnk);
   IConnectionSetPtr pConnectionsetPtr(pUnk);
   CCfString strName(L"HMI-Connection");
   CCfSmartString strName1(L"HMI-ConnectionS7Plus");
   CCfArrayVariant vtArrayVaiarnt;
   vtArrayVaiarnt.Append(strName);
   vtArrayVaiarnt.Append(strName1);
   ICfArrayVariantPtr connectionNames;
   vtArrayVaiarnt.DetachEnumerator(&connectionNames);
   pConnectionsetPtr->Add(connectionNames);
   int32_t count;
   pConnectionsetPtr->GetCount(&count);

   pConnectionsetPtr->Remove(strName1.AllocCFSTR());
   pConnectionsetPtr->GetCount(&count);

   pConnectionsetPtr->Clear();
   pConnectionsetPtr->GetCount(&count);
}
```

See also

IConnectionReadNotification (Page 8102)

IConnectionStateChangeNotification (Page 8103)

IConnection (Page 8101)

IConnectionResultEnumerator (Page 8096)

IConnectionStatusResultEnumerator (Page 8100)

20.3.9.6 Interfaces of the Plant Model

IPlantModel

Description

The C++ interface "IPlantModel" specifies methods for access to object instances of the plant model of a Runtime system. The "IPlantModel" object represents the plant model of the graphical Runtime system.

The interface inherits from the "ICfDispatch" interface.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[PlantObjectID/.../]PlantObjectID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following methods are specified in the interface:

"GetPlantObject" method

Supplies an "IPlantObject" instance.

```
CFRESULT GetPlantObject(const CFSTR Node, IPlantObject**
ppPlantObject)
```

- `Node`
  [in]: Identifies an `IPlantObject` instance by its name or its path in the hierarchy.
- `ppPlantObject`
  [out]: Points to an "IPlantObject" instance.

"GetPlantObjectsByType" method

Supplies an enumeration with "IPlantObject" instances that have a specific type.

```
GetPlantObjectsByType(const CFSTR plantObjectTypeFilter, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `plantObjectTypeFilter`
  [in]: Filter for the "IPlantObject" type on which the instances are based.
- `ViewFilter`
  [in]: Filter for the path in the hierarchy. Only instances from a specific node are returned.
- `ppPlantObjects`
  [out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

"GetObjectsByPropertyName" method

Supplies an enumeration with "IPlantObject" instances that have specific properties and originate in a specific plant node.

```
GetPlantObjectsByPropertyNames(const CFVARIANT PropertyNames, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `PropertyNames`
  [in]: Property names
- `ViewFilter`
  [in]: Filter for a hierarchy path.
- `ppPlantObjects`
  [out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

"GetPlantObjectsByExpression" method

Supplies an enumeration with "IPlantObject" instances. The instances are filtered by type and property values.

```
GetPlantObjectsByExpression(const CFVARIANT PropertyNames, const
CFSTR plantObjectTypeFilter, const CFSTR expressionFilter, const
CFSTR ViewFilter, IPlantObjectEnumerator** ppPlantObjects)
```

- `PropertyNames`
  [in]: Property names
  If the list contains multiple values, all properties must be available at the object.
- `plantObjectTypeFilter`
  [in]: Filter for the object type on which the instances are based.
- `expressionFilter`
  [in]: An expression that is a filter for the property values.
- `ViewFilter`
  Filter for a hierarchy path.
- `ppCpmNodes`
  [out]: Points to an "IPlantObjectEnumerator" enumeration with "IPlantObject" instances.

Example

Hierarchy path	Referenced object instance
`System2.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of system2.
`.TechnologicalHierarchy::P1/S1/L2/LeftPump`	References the "LeftPump" object instance in the "TechnologicalHierarchiy" of the local system.
`U4711`	References the "U4711" object instance of the local system.
`System2::U4711`	References the "U4711" object instance of System2.

Copy code

```
void PlantModelGetPlantObjectsByType(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectEnumeratorPtr pPlantObjectEnum;
   pPlantModel->GetPlantObjectsByType(CCfString(L"BLOWER").Get(), CCfString(L"").Get(),
&pPlantObjectEnum);
   IPlantObjectPtr pItem;
   pPlantObjectEnum->MoveNext();
   while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pItem)))
   {
      DisplayNodeInfo(pItem);
      pPlantObjectEnum->MoveNext();
   }



void PlantModelGetPlantObjectsByExpression(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectEnumeratorPtr pPlantObjectEnum;
   CCfString str1 = L"NumberOfNodes";
   CCfString str2 = L"Quality";
   CCfSafeArray daSafeArray(CF_VT_STR, 1U, 2U);
   int32_t index = 0;
   daSafeArray.PutElement(&index, (void*)&str1);
   ++index;
   daSafeArray.PutElement(&index, (void*)&str2);
   ++index;
   CCfVariant vtProperties;
   daSafeArray.Detach(&vtProperties);
   pPlantModel->GetPlantObjectsByExpression(vtProperties, CCfString(L"BLOWER").Get(),
CCfString(L"NumberOfNodes>=0"), CCfString(L"RUNTIME_1.hierarchy").Get(),
&pPlantObjectEnum);

   pPlantObjectEnum->MoveNext();
   IPlantObject* pPlantObject;
   while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pPlantObject)))
   {
      DisplayNodeInfo(pPlantObject);
      pPlantObjectEnum->MoveNext();
   }
}
```

IPlantObject

Description

The C++ interface "IPlantObject" specifies methods for handling object instances of the plant model of a Runtime system.

An object instance in the plant model is based on an object type and its data structure. Each object instance receives its position within the plant hierarchy by assigning it to a hierarchy node.

The interface inherits from the "ICfDispatch" interface.

Formatting of a hierarchy path

A hierarchy path of object instances consists of several components and has the following syntax:

```
[SystemName].HierarchyName::[NodeID/.../]NodeID
```

The system name can be omitted for referencing a local hierarchy. The dot before the hierarchy name must stay.

Members

The following methods are specified in the interface:

"GetName" method

Supplies the name of the "IPlantObject" instance for unique identification.

```
CFRESULT GetName(CFSTR* pName)
```

- `pName`
  [out]: Name of the instance

"GetParent" method

Supplies the parent of the "IPlantObject" instance in the hierarchy.

```
CFRESULT GetParent(IPlantObject** ppParent)
```

- `ppParent`
  [out]: The parent as "IPlantObject" instance.

"GetChildren" method

Supplies an enumeration with the children of the "IPlantObject" instance in the hierarchy.

```
CFRESULT GetChildren(IPlantObjectEnumerator** ppChildren)
```

- `ppChildren`
  [out]: An enumeration of the type "IPlantObjectEnumerator" with child object instances.

"GetPlantViewPaths" method

Supplies a map with hierarchy names and hierarchy paths that the "IPlantObject" instance has in all hierarchies in which it is included.

```
CFRESULT GetPlantViewPaths(ICfMapStringToVariant **pViewPaths)
```

- `pViewPaths`
  [out]: A map with String/String pairs (hierarchy name to hierarchy path).

"GetCurrentPlantView" method

Supplies the path and names of the "IPlantObject" instance in the current hierarchy.

If the "IPlantObject" instance is only contained in one hierarchy, this path is returned.

```
CFRESULT GetCurrentPlantView)(CFSTR* pView)
```

- `pView`
  [out]: Hierarchy path and name of the "IPlantObject" instance.

"SetCurrentPlantView" method

"CurrentPlantView" is the basis for navigation with the "GetParent" or "GetChildren" methods. If the "IPlantObject" instance is contained in several hierarchies, the path must be set via "SetCurrentPlantView" before the "GetParent" or "GetChildren" method can be used.

```
CFRESULT SetCurrentPlantView)(CFSTR const& View)
```

- `View`
  [in]: The current hierarchy

"GetProperty" method

Supplies a property of the "IPlantObject" instance.

```
CFRESULT GetProperty(const CFSTR propertyName,
IPlantObjectProperty** ppPlantObjectProperty)
```

- `propertyName`
  [in]: Name of a property of the "IPlantObject" instance
- `ppPlantObjectProperty`
  [out]: Points to an "IPlantObjectProperty" instance.

"GetProperties" method

Supplies a two-dimensional list (name-object pairs) of the data structure of the "IPlantObject" instance. The list allows access to the instance properties.

```
CFRESULT GetProperties(const CFVARIANT propertyNames,
IPlantObjectPropertySet** ppPlantObjectPropertySet)
```

- Optional: `propertyNames`
  [in]: List with names of one or multiple properties of the "IPlantObject" instance.
  Without "propertyNames" parameter, all properties of the instance are referenced in the list.
- `ppPlantObjectPropertySet`
  [out]: Points to the list of the type "IPlantObjectPropertySet" that contains the names of one or multiple properties of the "IPlantObject" instance.

"GetActiveAlarms" method

Supplies all active alarms that the "IPlantObject" instance contains at the time it is called in the active hierarchy. Unlike with an AlarmSubscription, no status changes or new alarms are signaled that occur after the function call. Users can filter the alarms or specify a SystemName if they only want to receive the active alarms of a specific system.

```
CFRESULT GetActiveAlarms(uint32_t language, CFBOOL IncludeChildren,
CFSTR filter,IN IPlantObjectAlarmCallback* pCallback)
```

- `language`
  [in]: Language code of the language for all texts of an alarm and the filters. Refer to the section Locale IDs of the supported languages (Page 7829).
- `IncludeChildren`
  [in]: The active alarms of the child instances are returned as well.
- `filter`
  [in]: SQL-type string for filtering the alarm texts. The filter can contain operators. See also Syntax of the alarm filter (Page 7828).
- `pCallback`
  [in]: Callback pointer.

"CreateAlarmSubscription" method

Supplies a "PlantObjectAlarmSubscription" that can be used to start and stop an alarm subscription.

```
CFRESULT CreateAlarmSubscription)(IPlantObjectAlarmSubscription**
ppPlantObjectAlarmSubscription)
```

- `ppPlantObjectAlarmSubscription`
  [out] Points to a "PlantObjectAlarmSubscription" instance.

Example

Copy code

```
void PlantObjectGetProperties(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectPtr pPlantObject;
   CCfString strNode = L".hierarchy::PlantView/Unit1";
   //gets node for specified Node path
   pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);

   //empty property - so all node members should be read
   CCfVariant vtProperties;
   IPlantObjectPropertySetPtr pPlantObjectPropertySet;
   // get the PlantObject properties by property names
   pPlantObject->GetProperties(vtProperties, &pPlantObjectPropertySet);

   uint32_t nCount = 0;
   pPlantObjectPropertySet->GetCount(&nCount);
   std::cout << "Count :" << nCount << std::endl;
}
```

IPlantObjectProperty

Description

The C++ interface "IPlantObjectProperty" specifies the handling of properties of object instances of the plant model of a Runtime system. The properties represent the data structure of an object instance.

The object instance communicates with the automation system through the properties of the data structure. The values of the properties are obtained from linked process tags or internal tags.

You reference an object using the `IPlantObject.GetProperty` or `IPlantObject.GetProperties` methods.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"GetName" method

Supplies the name of the property.

```
CFRESULT GetName(CFSTR *pName)
```

- `pName`
  [out]: Points to the name of the property.

"Read" method

Reads the value of the "IPlantObjectProperty" instance synchronously and returns it as an "IPlantObjectPropertyValue" object. The value, the quality code and the time stamp of the property are determined when the property is read.

```
CFRESULT Read(IPlantObjectPropertyValue**
ppPlantObjectPropertyValue)
```

- `ppPlantObjectPropertyValue`
  [out]: Values of the property as "IPlantObjectPropertyValue" instance

"Write" method

Writes the value synchronously to the "IPlantObjectProperty" instance.

```
CFRESULT Write)(const CFVARIANT value)
```

- `value`
  [in]: New process value of the property

Example

Copy code

```
void PlantObjectReadproperty(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectPtr pPlantObject;
   CCfString strNode = L".hierarchy::PlantView/Unit1";
   //gets node for specified Node path
   pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
   //empty property - so all node members should be read
   CCfVariant vtProperties;
   CCfSafeArray daPropertyNames(CF_VT_STR, 1U, 1U);
   int32_t index = 0;
   CCfString strName(L"DateActivation");
   daPropertyNames.PutElement(&index, &strName);
   daPropertyNames.Detach(&vtProperties);
   IPlantObjectPropertySetPtr pPlantObjectPropertySet;
   // get the PlantObject properties by property names
   pPlantObject->GetProperties(vtProperties, &pPlantObjectPropertySet);
   CCfString strName(L"NumberOfNodes");
   IPlantObjectPropertyPtr PlantObjectProperty;
   //get the PlantObject property by name
   pPlantObject->GetProperty(strName.Get(), &PlantObjectProperty);
   IPlantObjectPropertyValuePtr pPlantObjectPropertyValue;
   // Read PlantObject Property
   PlantObjectProperty->Read(&pPlantObjectPropertyValue);

   CCfString strName;
   pPlantObjectPropertyValue->GetPlantObjectPropertyName(&strName);
   std::cout << "PlantModelPropertyName:" << strName.ToUTF8() << std::endl;
   int64_t qc;
   pPlantObjectPropertyValue->GetQuality(&qc);
   std::cout << "Quality:" << qc << std::endl;
   int64_t ec;
   pPlantObjectPropertyValue->GetError(&ec);
   std::cout << "Error:" << ec << std::endl;
   CCfDateTime64 dt;
   pPlantObjectPropertyValue->GetTimeStamp(&dt);
   std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
   CCfVariant vtVal;
   pPlantObjectPropertyValue->GetValue(&vtVal);
   std::cout << "Value:" << vtVal.uint64 << std::endl;
}
```

IPlantObjectPropertyValue

Description

The C++ interface "IPlantObjectPropertyValue" specifies the handling of process tag properties of the Runtime system.

Members

The following methods are specified in the interface:

"GetPlantObjectPropertyName" method

Supplies the name of the tag.

```
CFRESULT GetPlantObjectPropertyName(CFSTR* pName)
```

- `pName`
  [out]: Points to the name.

"GetValue" method

Supplies the tag value.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- `pValue`
  [out]: Points to the process value.

"GetQuality" method

Supplies the quality code of the tag.

```
CFRESULT GetQuality(int64_t* pQualityCode)
```

- `pQualityCode`
  [out]: Points to the quality code.

"GetTimeStamp" method

Supplies the time stamp of the last modification to the tag.

```
CFRESULT GetTimeStamp(CFDATETIME64* pTimeStamp)
```

- `pTimeStamp`
  [out]: Points to the time stamp.

"GetError" method

Supplies the error code of the tag.

```
CFRESULT GetError(int64_t* pErrorCode)
```

- `pErrorCode`
  [out]: Points to the error code.

Example

For example, see IPlantObjectProperty (Page 8115).

IPlantModelPropertySubscriptionNotification

Description

The C++ interface "IPlantModelPropertySubscriptionNotification" defines methods for implementing asynchronous change monitoring of object instance properties. The methods are used by the C++ interface "ICpmNodePropertySet".

All the methods return CF_SUCCESS following successful execution.

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"OnDataChanged" method

Callback method is called when a monitored object instance property is changed.

```
CFRESULT OnDataChanged(IPlantObjectPropertyValueEnumerator*
pEnumerator)
```

- `pEnumerator`:
  [in]: Points to the "IPlantObjectPropertyValueEnumerator" object that provides access to an enumeration of "IPlantObjectPropertyValue" instances.

Example

Register a PropertySet for monitoring:

Copy code

```
void PlantObjectSubscribePropertySet(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);

   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectPtr pPlantObject;
   CCfString strNode = L".hierarchy::PlantView/Unit1/Filler1";
   //gets node for specified Node path
   pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);

   //empty property - so all node members should be read
   CCfVariant vtProperties;
   IPlantObjectPropertySetPtr pPlantObjectPropertySet;
   // get the PlantObject properties by property names
   pPlantObject->GetProperties(vtProperties, &pPlantObjectPropertySet);

   CPlantModelPropertySubscriptionNotification* pSubscribe = new
CPlantModelPropertySubscriptionNotification();
   IPlantModelPropertySubscriptionNotification* pNotification = pSubscribe;
   //Susbcribe for all PlantObject properties
   pPlantObjectPropertySet->Subscribe(pNotification);
}



CFRESULT CFCALLTYPE
CPlantModelPropertySubscriptionNotification::OnDataChanged(IPlantObjectPropertyValueEnumer
ator* pEnumerator)
{
   uint32_t nCount = 0;
   pEnumerator->Count(&nCount);
   for (int i = 0;i < (int32_t)nCount; i++)
   {
      pEnumerator->MoveNext();
      IPlantObjectPropertyValue* pPlantModelPropertyValue;
      if (CF_SUCCEEDED(pEnumerator->Current(&pPlantModelPropertyValue)) && nullptr !=
pPlantModelPropertyValue)
      {
         CCfString strName;
         pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
         std::cout << "Name:" << strName.ToUTF8() << std::endl;
         int64_t qc;
         pPlantModelPropertyValue->GetQuality(&qc);
         std::cout << "Quality:" << qc << std::endl;
         int64_t ec;
         pPlantModelPropertyValue->GetError(&ec);
         std::cout << "Error:" << ec << std::endl;
         CCfDateTime64 dt;
         pPlantModelPropertyValue->GetTimeStamp(&dt);
         std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
         CCfVariant vtVal;
         pPlantModelPropertyValue->GetValue(&vtVal);
         std::cout << "Value:" << vtVal.uint64 << std::endl;
```

Copy code

```
        }
    }
    this->SetEvent();
    return CF_SUCCESS;
}
```

IPlantObjectPropertyValueEnumerator

Description

The C++ interface "IPlantObjectPropertyValueEnumerator" specifies methods for handling the enumeration of "IPlantObjectPropertyValue" instances.

All the methods return `CF_SUCCESS` following successful execution.

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"Current" method

Supplies the current element of the enumeration.

```
Current(IPlantObjectPropertyValue** ppItem)
```

- `ppItem`
  [out]:

"MoveNext" method

Move to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Reset" method

Resets the current position in the enumeration to the first element.

```
CFRESULT Reset()
```

"MoveNext" then supplies the first element of the enumeration.

"Count" method

Output the size of the enumeration or the number of elements in an enumeration.

```
CFRESULT Count(uint32_t *pValue)
```

- `value`
  [out]: Points to the value for the number of elements in the enumeration.

Example

For example, see IPlantModelPropertySubscriptionNotification (Page 8118).

IPlantObjectPropertySet

Description

The C++ interface "IPlantObjectPropertySet" specifies methods for optimized access to several "IPlantObjectProperty" instances of an "IPlantObject" instance of the Runtime system.

After initialization of the "IPlantObjectPropertySet" object, you have read/write access to multiple "IPlantObjectProperty" instances in one call. Simultaneous access has better performance and a lower communication load than single access to multiple properties.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"GetContextID" method

Supplies an ID as additional identification feature of a property. "ContextId" can, for example, be used to recognize properties having the same name but from different systems.

Default value -1: "ContextId" is not used.

```
CFRESULT GetContextId)(int32_t* pContextId)
```

- `pContextId`
  [out]: "ContextId" of the property

"SetContextID" method

Sets an ID as additional identification feature of a property. "ContextId" can, for example, be used to recognize properties having the same name but from different systems.

Default value -1: "ContextId" is not used.

```
CFRESULT SetContextId(int32_t contextId)
```

- `contextId`
  [in]: "ContextId" of the property

"Read" method

Supplies all values of the "IPlantObjectProperty" instances contained in the "IPlantObjectPropertySet" instance. The values are read synchronously.

```
CFRESULT Read(IPlantObjectPropertyValueEnumerator**
ppPlantObjectPropertyValueEnumerator)
```

- `ppPlantObjectPropertyValueEnumerator`
  [out]: Points to the enumeration of the tag values as an "IPlantObjectPropertyValueEnumerator" object.

"ReadAsync" method

Reads the values of all "IPlantObjectProperty" instances of the "IPlantObjectPropertySet" instance asynchronously.

```
CFRESULT ReadAsync(IPlantObjectPropertySetReadReply* pReadReply)
```

- `pReadReply`
  [in]: Points to the "IPlantObjectPropertySetReadReply" object that implements the callback interface for read operations.

"Write" method

Writes the values of the "IPlantProperty" instances of the "PlantObjectPropertySet" instance synchronously to the Runtime system.

```
CFRESULT Write(HmiUnified::Rt::IErrorResultEnumerator**
ppEnumerator)
```

- `pWriteReply`
  [out]: Points to an "IErrorResultEnumerator" object that contains the enumeration with errors for the write operations of the "IPlantObjectProperty" instances.

"WriteAsync" method

Writes the values of all "IPlantObjectProperty" instances of the "PlantObjectPropertySet" instance asynchronously to the Runtime system.

```
CFRESULT WriteAsync(IPlantObjectPropertySetWriteReply *pWriteReply)
```

- `pWriteReply`
  [in]: Points to the "IPlantObjectPropertySetWriteReply" object that implements the callback interface for write operations.

"GetCount" method

Supplies the number of properties of the "IPlantObjectPropertySet" instance.

```
CFRESULT GetCount(uint32_t *pCount)
```

- `pCount`
  [out]: Points to the number of properties.

"Subscribe" method

Subscribes all properties of the "IPlantObjectPropertySet" instance asynchronously for change monitoring.

```
CFRESULT Subscribe(IPlantModelPropertySubscriptionNotification*
pPlantModelSubscriptionCallback)
```

- `pPlantModelSubscriptionCallback`
  [in]: Points to the "IPlantObjectPropertySubscriptionNotification" object that implements the callback interface of the change monitoring.

"CancelSubscribe" method

Cancels change monitoring for all properties of the "IPlantObjectPropertySet" instance.

```
CFRESULT CancelSubscribe()
```

"Add" method

Adds an "IPlantObjectProperty" instance with property value or more "IPlantObjectProperty" instances to the "IPlantObjectPropertySet" instance.

```
CFRESULT Add(ICfArrayVariantPtr propertyNames)
```

- `propertyNames`
  [in]: Array with names of several "IPlantObjectProperty" instances

or

```
CFRESULT Add(const CFSTR propertyName, const CFVARIANT value)
```

- `propertyName`
  [in]: Name of the "IPlantObjectProperty" instance
- `value`
  [in]: New process value of the "IPlantObjectProperty" instance

"Remove" method

Removes a property from the "IPlantObjectPropertySet" instance.

```
CFRESULT Remove(const CFSTR propertyName)
```

- `propertyName`
  [in]: Name of the property that is being removed.

"Clear" method

Removes all properties from the "IPlantObjectPropertySet" instance.

```
CFRESULT Clear()
```

New example

Copy code

```
void PlantObjectWriteAsyncPropertySet(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectPtr pPlantObject;
   CCfString strNode = L".hierarchy::PlantView/Unit1";
   //gets node for specified Node path
   pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);

   //empty property - so all node members should be read
   CCfVariant vtProperties;
   IPlantObjectPropertySetPtr pPlantObjectPropertySet;
   // get the PlantObject properties by property names
   pPlantObject->GetProperties(vtProperties, &pPlantObjectPropertySet);

   CCfString strName(L"NumberOfNodes");
   CCfVariant vtValue(1000);
   pPlantObjectPropertySet->Add(strName, vtValue);

   CPlantObjectPropertySetWriteReply* pReply = new CPlantObjectPropertySetWriteReply();
   IPlantObjectPropertySetWriteReplyPtr pWritReply = pReply;
   //Write PlantObject properties values asynchronously
   pPlantObjectPropertySet->WriteAsync(pWritReply);
}
```

IPlantObjectPropertySetReadReply

Description

The C++ interface "IPlantObjectPropertySetReadReply" defines the "OnReadComplete" method as callback method of a "ReadAsync" call. The method is used by the C++ interface "IPlantObjectPropertySet".

Members

The following methods are specified in the interface:

"OnReadComplete" method

The "OnReadComplete" callback method is called when the "ReadAsync" method is used.

```
CFRESULT OnReadComplete(CFVARIANT
systemError, IPlantObjectPropertyValueEnumerator*
pPlantObjectPropertyValueEnumerator)
```

- `systemError`
  [in]: Supplies an error code when a global error has occurred.
- `ppPlantObjectPropertyValueEnumerator`
  [in]: Points to an "IPlantObjectPropertyValueEnumerator" object that contains the enumeration of the process values of object instance properties.

Example

Copy code

```
void PlantObjectReadAsyncPropertySet(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectPtr pPlantObject;
   CCfString strNode = L".hierarchy::PlantView/Unit1";
   //gets node for specified Node path
   errCode = pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);

   //empty property - so all node members should be read
   CCfVariant vtProperties;
   IPlantObjectPropertySetPtr pPlantObjectPropertySet;
   // get the PlantObject properties by property names
   pPlantObject->GetProperties(vtProperties, &pPlantObjectPropertySet);
   CPlantObjectPropertySetReadReply* pReply = new CPlantObjectPropertySetReadReply();
   IPlantObjectPropertySetReadReplyPtr pReadReply = pReply;
   // Read PlantObject properties values asynchronously
   pPlantObjectPropertySet->ReadAsync(pReadReply);
}
CFRESULT CFCALLTYPE CPlantObjectPropertySetReadReply::OnReadComplete(IN CFVARIANT
SystemError, IN IPlantObjectPropertyValueEnumerator* pPlantObjectPropertySet)
{
   uint32_t nCount = 0;
   pPlantObjectPropertySet->Count(&nCount);

   for (int i = 0;i < (int32_t)nCount; i++)
   {
      pPlantObjectPropertySet->MoveNext();
      IPlantObjectPropertyValue* pPlantModelPropertyValue;

      CCfString strName;
      pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
      std::cout << "Name:" << strName.ToUTF8() << std::endl;
      int64_t qc;
      pPlantModelPropertyValue->GetQuality(&qc);
      std::cout << "Quality:" << qc << std::endl;
      int64_t ec;
      pPlantModelPropertyValue->GetError(&ec);
      std::cout << "Error:" << ec << std::endl;
      CCfDateTime64 dt;
      pPlantModelPropertyValue->GetTimeStamp(&dt);
      std::cout << "DateTime:" << dt.GetDateTimeString().ToUTF8() << std::endl;
      CCfVariant vtVal;
      pPlantModelPropertyValue->GetValue(&vtVal);
      std::cout << "Value:" << vtVal.uint64 << std::endl;
   }
   this->SetEvent();
   return CF_SUCCESS;
}
```

IPlantObjectPropertySetWriteReply

Description

The C++ interface "IPlantObjectPropertySetWriteReply" defines the "OnWriteComplete" method as callback method of a "WritedAsync" call. The method is used by the C++ interface "IPlantObjectPropertySet".

Members

The following methods are specified in the interface:

"OnWriteComplete" method

The "OnWriteComplete" callback method is called when the "WriteAsync" method is used.

```
CFRESULT OnWriteComplete(CFVARIANT
systemError, IPlantObjectPropertyValueEnumerator*
pPlantObjectPropertyValueEnumerator)P
```

- `systemError`
  [in]: Supplies an error code when a global error has occurred.
- `ppPlantObjectPropertyValueEnumerator`
  [in]: Points to an "IPlantObjectPropertyValueEnumerator" object that contains the enumeration of the process values of "IPlantObjectProperty" instances.

Example

Copy code

```
CFRESULT CFCALLTYPE CPlantObjectPropertySetWriteReply::OnWriteComplete(IN CFVARIANT
SystemError, IN IPlantObjectPropertyValueEnumerator* pPlantObjectPropertySet)
{
   uint32_t nCount = 0;
   pPlantObjectPropertySet->Count(&nCount);

   for (int i = 0;i < (int32_t)nCount; i++)
   {
      pPlantObjectPropertySet->MoveNext();
      IPlantObjectPropertyValue* pPlantModelPropertyValue;
      CCfString strName;
      pPlantModelPropertyValue->GetPlantObjectPropertyName(&strName);
      std::cout << "Name:" << strName.ToUTF8() << std::endl;
      int64_t ec;
      pPlantModelPropertyValue->GetError(&ec);
      std::cout << "Error:" << ec << std::endl;
   }

   this->SetEvent();
   return CF_SUCCESS;
}
```

IPlantObjectEnumerator

Description

The "IPlantObjectEnumerator" interface is a C++ interface that specifies methods for handling the enumeration of object instances of the plant model of a Runtime system.

All the methods return CF_SUCCESS following successful execution.

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"Current" method

Supplies the current element of the enumeration.

```
CFRESULT Current(IPlantObject **ppItem)
```

- `ppItem`
  [out]: Points to the current "IPlantObject" object as an element of the list.

"MoveNext" method

Move to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Reset" method

Resets the current position in the enumeration to the first element.

```
CFRESULT Reset()
```

"MoveNext" then supplies the first element of the enumeration.

"Count" method

Output the size of the enumeration or the number of elements in an enumeration.

```
CFRESULT Count(uint32_t *pValue)
```

- `value`
  [out]: Points to the value for the number of elements in the enumeration.

Example

Copy code

```
void PlantObjectCurrentViewPath(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel"), &pUnk);
   IPlantModelPtr pPlantModel(pUnk);
   IPlantObjectEnumeratorPtr pPlantObjectEnum;
   CCfString str1 = L"NumberOfNodes";
   CCfString str2 = L"Quality";
   CCfString str3 = L"Quantity";
   CCfString str4 = L"DateActivation";
   CCfSafeArray daSafeArray(CF_VT_STR, 1U, 4U);
   int32_t index = 0;
   daSafeArray.PutElement(&index, (void*)&str1);
   ++index;
   daSafeArray.PutElement(&index, (void*)&str2);
   ++index;
   daSafeArray.PutElement(&index, (void*)&str3);
   ++index;
   daSafeArray.PutElement(&index, (void*)&str4);
   CCfVariant vtProperties;
   daSafeArray.Detach(&vtProperties);
   // gets PlantObjects for specified filter
   pPlantModel->GetPlantObjectsByExpression(vtProperties,
CCfString(L"UNIT").Get(), CCfString(L""),
CCfString(L"RUNTIME_1.hierarchy::PlantView/Unit1").Get(),
&pPlantObjectEnum);

   pPlantObjectEnum->MoveNext();
   IPlantObject* pPlantObject;
   while (CF_SUCCEEDED(pPlantObjectEnum->Current(&pPlantObject)))
   {
      DisplayNodeInfo(pPlantObject);
      pPlantObjectEnum->MoveNext();
   }
}
```

IPlantObjectAlarmSubscription

Description

The C++ interface "IPlantObjectAlarmSubscription" specifies methods for starting and stopping an "AlarmSubscription".

The interface inherits from the "ICfDispatch" interface.

Members

The following methods are specified in the interface:

"Start" method

Subscribe systems for monitoring changes of active alarms.

```
CFRESULT Start(IPlantObjectAlarmSubscriptionCallback* callbackPtr)
```

- `callbackPtr`
  [in]: Points to the "IPlantObjectAlarmSubscriptionCallback" object that implements the callback interface of the change monitoring.

"Stop" method

Clear monitoring of active alarms.

```
CFRESULT Stop(void)
```

"GetFilter" method

Supplies the string by which the result set is filtered.

```
CFRESULT GetFilter(CFSTR* filter)
```

- `filter`
  [out]: SQL-type string for filtering the result set of active alarms.

"SetFilter" method

Sets the string for filtering the result set of active alarms.

```
CFRESULT SetFilter(IN CFSTR filter)
```

- `filter`
  [in]: SQL-type string for filtering the result set of active alarms.
  All properties of an alarm can be used in the filter string. The filter string can contain operators. Refer to the section Syntax of the alarm filter (Page 7828).

"GetLanguage" method

Supplies the country identifier of the language of the monitored alarms.

```
CFRESULT GetLanguage(uint32_t* language)
```

- `language`
  [out]: Code of the country identification. See also sectionLocale IDs of the supported languages (Page 7829).

"SetLanguage" method

Sets the country identifier of the language of the monitored alarms.

```
CFRESULT SetLanguage)(uint32_t language)
```

- `language`
  [in]: Code of the country identification See also sectionLocale IDs of the supported languages (Page 7829).

"GetIncludeChildren" method

Supplies the setting for the child instances.

```
CFRESULT GetIncludeChildren(CFBOOL* bIsIncludeChildren)
```

- `bIsIncludeChildren`
  [out]: Reads out whether the child instances are part of monitoring.

"SetIncludeChildren" method

Determines the setting for the child instances.

```
CFRESULT SetIncludeChildren(CFBOOL bIsIncludeChildren)
```

- `bIsIncludeChildren`
  [in]: Controls whether the child instances are part of monitoring.

Example

Copy code

```
void PlantObjectSubscription(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"PlantModel").Get(), &pUnk);

   IPlantModelPtr pPlantModel(pUnk);
   CPlantModelAlarmValue* pAlarmValue = new CPlantModelAlarmValue();
   pAlarmValue->AddRef();
   IPlantObjectAlarmCallback *pCB = pAlarmValue;
   CCfString strNode = L".hierarchy::RootNodeName/Node1";
   IPlantObjectPtr pPlantObject;
   pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
   CCfSmartString daFilter;//= L"AlarmClassName = 'Alarm'";
   uint32_t daLanguage = 1033;
   IPlantObjectAlarmSubscriptionPtr pPlantObjectAlarmSubscription;
   pPlantObject->CreateAlarmSubscription(&pPlantObjectAlarmSubscription);
   pPlantObjectAlarmSubscription->SetFilter(daFilter.AllocCFSTR());
   pPlantObjectAlarmSubscription->SetLanguage(1033);
   pPlantObjectAlarmSubscription->SetFilter(false);
   pPlantObjectAlarmSubscription->SetIncludeChildren(false);
   CCfRefPtr<CPlantObjectAlarmSubscriptionCallback> pCallback = new
CPlantObjectAlarmSubscriptionCallback();
   pPlantObjectAlarmSubscription->Start(pCallback);
   if (0 == pCallback->WaitForcompletion(30000))
   {
      pPlantObjectAlarmSubscription->Stop();
   }
}
```

IPlantObjectAlarmCallback

Description

The C++ interface "IPlantObjectAlarmCallback" defines the callback method "OnAlarm".

Member

The following methods are defined in the interface:

"OnAlarm" method

Specifies the signature of the event handling method for the "OnAlarm" event of an "IPlantObject" instance.

```
OnAlarm(uint32_t systemError,
        CFSTR systemName,

Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator*
pItems,
        CFBOOL completed)
```

- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `pItems` is irrelevant.
- `systemName`
  Name of the runtime system that is subscribed for alarm monitoring by the user.
- `pItems`
  Supplies a pointer to "IAlarmResultEnumerator" that can be used to enumerate the active alarms.
- `completed`
  Status of the asynchronous transfer:
  - `True`: All alarms are read out.
  - `False`: Not all alarms are read out yet.

Example

Copy code

```
// Callback for alarm notifications
CFRESULT CFCALLTYPE CPlantModelAlarmValue::OnAlarm( uint32_t systemError, CFSTR
systemName, Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator* pItems, CFBOOL
completed)
{
   CFRESULT hr = CF_FALSE;
   if (!completed)
   {
      uint32_t nsize;
      pItems->Count(&nsize);
      if (nsize > 0 && CF_SUCCEEDED(systemError)) {
      {
         int index = 0;
         while (CF_SUCCEEDED(pItems->MoveNext()))
         {
            IAlarmResultPtr ppValues;
            pItems->Current(&ppValues);

            CCfString strId;
            ppValues->GetSourceID(&strId);
            std::cout << "String ID = " << strId.ToUTF8().c_str() << std::endl;
            CCfString strName;
            ppValues->GetName(&strName);
            std::cout << "Name = " << strName.ToUTF8().c_str() << std::endl;
            CCfString strClassName;
            ppValues->GetAlarmClassName(&strClassName);
           std::cout << "Alarm Class Name = " << strClassName.ToUTF8().c_str() << std::endl;
            index++;
         }
      }
   }
   hr = CF_SUCCESS;
   return hr;
}
```

IPlantObjectAlarmSubscriptionCallback

Description

The C++ interface "IPlantObjectAlarmCallbackSubscription" defines the callback method "OnAlarm".

The interface inherits the "ICfUnknown" interface.

Members

The following methods are specified in the interface:

"OnAlarm" method

Specifies the signature of the event handling method for the "OnAlarm" event of an "IPlantObject" instance.

```
OnAlarm(uint32_t systemError,
        CFSTR systemName,

Siemens::Runtime::HmiUnified::Alarms::IAlarmResultEnumerator*
pItems)
```

- `systemError`
  Supplies an error code when a global error has occurred. When the error code is set, `pItems` is irrelevant.
- `systemName`
  Name of the runtime system that is subscribed for alarm monitoring by the user.
- `pItems`
  - Supplies a pointer to "IAlarmResultEnumerator" that can be used to enumerate the active alarms.

20.3.9.7 Interfaces of the Calendar option

ISHCCalendarOption

Description

The C++ interface "ISHCCalendarOption" specifies the "GetObject" method. The method supplies the calendar object of an "IPlantObject" instance. A calendar is always integrated via an "IPlantObject" instance.

Members

"GetObject" method

Supplies an error code of the type `CFRESULT`.

```
CFRESULT
GetObject(Siemens::Runtime::HmiUnified::PlantModel::IPlantObject *
pCpmNode, CFSTR objectName, ICfUnknown** ppObject)
```

- `cpmNode`
  [in]: Reference to the "IPlantObject" instance currently selected in the hierarchy
- `objectName`
  [in]: The name of the "IPlantObject" instance
- `ppObject`
  [out]: Returns an object of the type "ICfUnknown". The object is cast to a calendar object using the "QueryInterface" method.
  Example:

```
ISHCCalendarOptionPtr pShcOption;
pRuntime->GetOption(CCfString(ODK_SHC_OPTION),
(IOption**)&pShcOption);
ICfUnknownPtr pUnk;
pShcOption->GetObject(pPlantObject, ODK_SHC_CALENDAR, &pUnk);
ISHCCalendarPtr pCal;
pUnk->QueryInterface(IID_ISHCCalendar, (ICfUnknown**)&pCal);
COdkShcSample Sample;
Sample.SetCalendar(pCal);
```

Example

The following example serves as a basis for the other examples for the C++ interfaces of the Calendar option.

It shows how you can obtain the "IPlantObject" instance and also an "ISHCCalendar" instance. The "ISHCCalendar" instance referenced via `pCalendar` is also used in the other examples.

Copy code

```
#include <iostream>
#include "IOdkShcInterface.h"
#include "IOdkRt.h"
#include "IOdkRtAlarm.h"
using namespace Siemens::Runtime::HmiUnified::Rt;
using namespace Siemens::Runtime::HmiUnified::Common;
using namespace Siemens::Runtime::HmiUnified::PlantModel;
IRuntimePtr pRuntime;
ISHCCalendarPtr pCalendar;
ISHCCalendarSettingsProviderPtr pCalendarProvider;
ISHCCategoryProviderPtr pShcCategoryProvider;
ISHCShiftTemplatesProviderPtr pShcShiftTemplateProvider;
ISHCDayProviderPtr pShcDayProvider;
ISHCDayTemplatesProviderPtr pShcDayTemplateProvider;
ISHCActionTemplatesProviderPtr pShcActionTemplateProvider;
CCfString projectName = L"";
if (CF_SUCCEEDED(Connect(projectName, &pRuntime)))
{
     ICfUnknownPtr pPlantModelUnk;
     pRuntime->GetObject(CCfString(L"PlantModel"), &pPlantModelUnk);
     IPlantObjectPtr pPlantObject;
     IPlantModelPtr pPlantModel(pPlantModelUnk);
     CCfString strNode = L".hierarchy::Plant/Unit1";
     //gets Object for specified Node path
     pPlantModel->GetPlantObject(strNode.Get(), &pPlantObject);
     IOptionPtr pOdkOption;
     pRuntime->GetOption(CCfString(ODK_SHC_OPTION), &pOdkOption);
     ISHCCalendarOptionPtr pShcOption;
     pOdkOption->QueryInterface(IID_ISHCCalendarOption, (ICfUnknown**)&pShcOption);
     ICfUnknownPtr pUnk;
     pShcOption->GetObject(pPlantObject, ODK_SHC_CALENDAR, &pUnk);
     pUnk->QueryInterface(IID_ISHCCalendar, (ICfUnknown**)&pCalendar);
     // Get all data provider
     pCalendar->GetActionTemplatesProvider(&pShcActionTemplateProvider);
     pCalendar->GetCategoryProvider(&pShcCategoryProvider);
     pCalendar->GetDayProvider(&pShcDayProvider);
     pCalendar->GetDayTemplateProvider(&pShcDayTemplateProvider);
     pCalendar->GetShiftTemplateProvider(&pShcShiftTemplateProvider);
     pCalendar->GetSettings(&pCalendarProvider);
}
```

ISHCCalendar

Description

The C++ interface "ISHCCalendar" specifies the methods of a calendar.

The interface inherits from the "ICfUnknown" interface.

Members

"GetSettings" method

Supplies the "ISHCCalendarSettings" instances of the calendar.

```
CFRESULT GetSettings(ISHCCalendarSettings** calendar)
```

- `calendar`
  [out]: The "ISHCCalendarSettings" instance

"GetCategoryProvider" method

Supplies an "ISHCCategoryProvider" instance. The provider enables access to the "ISHCategory" instances of the calendar.

```
CFRESULT GetCategoryProvider(ISHCCategoryProvider**
ppCatgoryProvider)
```

- `ppCatgoryProvider`
  [out]: The "ISHCCategoryProvider" instance

"GetShiftTemplateProvider" method

Supplies an "ISHCShiftTemplatesProvider" instance. The provider enables access to the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT GetShiftTemplateProvider(ISHCShiftTemplatesProvider**
ppShiftTemplateProvider)
```

- `ppShiftTemplateProvider`
  [out]: The "ISHCShiftTemplatesProvider" instance

"GetDayTemplateProvider" method

Supplies an "ISHCDayTemplatesProvider" instance. The provider enables access to the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT GetDayTemplateProvider(ISHCDayTemplatesProvider**
ppDayTemplateProvider)
```

- `ppDayTemplateProvider`
  [out]: The "ISHCDayTemplatesProvider" instance

"GetActionTemplatesProvider" method

Supplies an "ISHCActionTemplatesProvider" instance. The provider enables access to the "ISHCActionTemplate" instances of the calendar.

```
CFRESULT GetActionTemplatesProvider)(OUT
ISHCActionTemplatesProvider** ppActionTemplatesProvider)
```

- `ppActionTemplatesProvider`
  [out]: The "ISHCActionTemplatesProvider" instance

"GetDayProvider" method

Supplies an "ISHCDayProvider" instance. The provider enables access to the "ISHCDay" instances of the calendar.

```
CFRESULT GetDayProvider(ISHCDayProvider** ppDayProvider)
```

- `ppDayProvider`
  [out]: The "ISHCDayProvider" instance

"GetObject" method

Creates an instance of the type defined in `value` and supplies a reference to the instance.

```
CFRESULT GetObject(const CFSTR value, ICfUnknown** ppObject)
```

- `value`
  [in]: Possible values:
  - "ODK_SHC_OPTION"
  - "ODK_SHC_CALENDAR"
  - "ODK_SHC_TIME_SLICE"
  - "ODK_SHC_DAY_TEMPLATE"
  - "ODK_SHC_DAY"
  - "ODK_SHC_DAY_TEMPLATE"
  - "ODK_SHC_ACTION_TEMPLATE"
  - "ODK_SHC_ACTION_TEMPLATE_ELEMENT"
- `ppObject`
  [out]: Reference to an object of the type "ICfUnknown", which can be cast to the corresponding type.
  Example:

```
ICfUnknownPtr pUnk;
ISHCShiftTemplatePtr pShcShiftTemplate;
pCalendar->GetObject(ODK_SHC_SHIFT_TEMPLATE, &pUnk);
pShcShiftTemplate = (ISHCShiftTemplatePtr)pUnk;
```

ISHCCalendarSettings

Description

The C++ interface "ISHCCalendarSettings" specifies methods for the calendar settings of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetPlantObject" method

Supplies the name of the "IPlantObject" instance to which the calendar belongs.

```
GetPlantObject(CFSTR* pCpmNode)
```

- `pCpmNode`
  [out]: The name of the "IPlantObject" instance

"GetFirstDayOfWeek" method

Supplies the first day of the week.

```
CFRESULT GetFirstDayOfWeek(CFENUM * pvarRet)
```

- `pvarRet`
  Points to the enumeration "ShcWeekDay", which can contain the following values:
  - Sunday (0)
  - Monday (1)
  - Tuesday (2)
  - Wednesday (3)
  - Thursday (4)
  - Friday (5)
  - Saturday (6)

"GetFirstWeekOfYear" method

Supplies the first week of the year.

```
CFRESULT GetFirstWeekOfYear(CFENUM * pvarRet)
```

- `pvarRet`
  [out]: Points to the enumeration "ShcWeekStart", which can contain the following values:
  - FirstOfJanuary (0): The first week starts on the first of January.
  - AtLeastFourDays (1): The first week must have at least four days.
  - WholeWeek (2): The first week must have at least seven days.

"GetFiscalYearStartDay" method

Supplies the first day of the fiscal year.

```
CFRESULT GetFiscalYearStartDay(uint8_t* pvarRet)
```

- `pvarRet`
  [out]: The first day of the fiscal year of the calendar.

"GetFiscalYearStartMonth" method

Supplies the first month of the fiscal year.

```
CFRESULT GetFiscalYearStartMonth(uint8_t* pvarRet)
```

- `pvarRet`
  [out]: The first month of the fiscal year of the calendar.

"GetDayOffset" method

Supplies the offset with which the workday begins, calculated from midnight.

```
CFRESULT GetDayOffset(CFTIMESPAN64 * pvarRet)
```

- `pvarRet`
  [out]: Supplies the number of hours after midnight with which the day begins.

"GetWorkDays" method

Supplies the workdays of the calendar.

```
CFRESULT GetWorkDays(uint8_t* pvarRet)
```

- `pvarRet`
  [out]: The workdays of the calendar.

"GetTimeZone" method

Supplies the time zone of the calendar.

```
CFRESULT GetTimeZone(uint32_t* pTimeZone)
```

- `pvarRet`
  [out]: The time zone ID of Microsoft.

ISHCCategory

Description

The C++ interface "ISHCCategory" specifies the methods of a time category of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCCategory" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- `pvarRet`
  [out]: The name

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCCategory" instance and their language code IDs.

```
CFRESULT GetDescriptions(OUT ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for description).

Example:

```
ICfMapIDToVariantPtr pDescriptions;
hr = pShcCategory->GetDescriptions(&pDescriptions);
std::cout << "Descriptions::" << std::endl << std::endl;
uint32_t nCount2 = 0;
pDescriptions->Count(&nCount2);
for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
{
   int32_t nLanguageID;
```

```
   pDescriptions->KeyAt(nIndex2, &nLanguageID);
   CCfVariant strDescription;
   pDescriptions->ValueAt(nLanguageID, &strDescription);
   std::cout << "LangauageID =" << nLanguageID << " Description="
<< CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
}
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCCategory" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for display name).

Example: Similar to "GetDescriptions"

"GetColor" method

Supplies the color of the "ISHCCategory" instance.

```
CFRESULT GetColor(uint32_t* pColor)
```

- `pColor`
  [out]: Returns a 4-byte value for an RGBA color value.

"GetIsDeleted" method

Supplies the information on whether the "ISHCCategory" instance was deleted in Engineering.

```
CFRESULT GetIsDeleted(CFBOOL* p_bIsDeleted) = 0;
```

- `p_bIsDeleted`
  [out]:
  - 0: Was not deleted (default)
  - 1: Was deleted

See also

Locale IDs of the supported languages (Page 7829)

ISHCCategoryEnumerator

Description

The C++ interface "ISHCCategoryEnumerator" specifies methods for handling the enumeration of the time categories of an "ISHCCalendar" instance. The enumeration is returned by the `Read` method of an "ISHCCategoryProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCCategory** ppItem)
```

- `ppItem`
  [out]: The current "ISHCCategory" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of categories

Example

Copy code

```
void PrintCategory(const ISHCCategoryEnumeratorPtr& p_pShcCategoryEnum)
{
   uint32_t nCount = 0;
   p_pShcCategoryEnum->Count(&nCount);
   for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
   {
      p_pShcCategoryEnum->MoveNext();
      ISHCCategoryPtr pShcCategory;
      p_pShcCategoryEnum->Current(&pShcCategory);
      cout << endl;
      CFRESULT hr = CF_ERROR;
      CCfString strName;
      hr = pShcCategory->GetName(&strName);
      cout << "CategoryName= " << strName.ToUTF8().c_str() << endl;
      uint32_t nColor;
      hr = pShcCategory->GetColor(&nColor);
      cout << "Color= " << nColor << endl;
      CFBOOL bIsDeleted;
      hr = pShcCategory->GetIsDeleted(&bIsDeleted);
      cout << "IsDeleted= " << (uint32_t)bIsDeleted << endl;

      ICfMapIDToVariantPtr pDisplayNames;
      hr = pShcCategory->GetDisplayNames(&pDisplayNames);
      std::cout << "DisplayNames::" << std::endl << std::endl;
      uint32_t nCount1 = 0;
      pDisplayNames->Count(&nCount1);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
      {
         int32_t nLanguageID;
         pDisplayNames->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strDIsplayname;
         pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
         std::cout << "LangauageID =" << nLanguageID << " DisplayName =" <<
CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
      }

      ICfMapIDToVariantPtr pDescriptions;
      hr = pShcCategory->GetDescriptions(&pDescriptions);
      std::cout << "Descriptions::" << std::endl << std::endl;
      uint32_t nCount2 = 0;
      pDescriptions->Count(&nCount2);
      for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
      {
         int32_t nLanguageID;
         pDescriptions->KeyAt(nIndex2, &nLanguageID);
         CCfVariant strDescription;
         pDescriptions->ValueAt(nLanguageID, &strDescription);
         std::cout << "LangauageID =" << nLanguageID << " Description=" <<
CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
      }
   }
}
```

ISHCCategoryProvider

Description

The C++ interface "ISHCCategoryProvider" provides you with read access to an "ISHCCategoryEnumerator" instance which contains an enumeration with the time categories of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCCategoryEnumerator" instance which has access to an enumeration with the "ISHCCategory" instances of the calendar.

```
CRFESULT Browse(ISHCCategoryEnumerator** data)
```

- `data`
  [out]: The enumerator

Example:

```
ISHCCategoryEnumeratorPtr pShcCategoryEnum;
CFRESULT hr = pShcCategoryProvider->Browse(&pShcCategoryEnum);
if (CF_SUCCEEDED(hr))
    PrintCategory(pShcCategoryEnum);
```

ISHCTimeSlice

Description

The C++ interface "ISHCTimeSlice" specifies the methods of a time slice.

The interface inherits from the "ICfUnknown" interface.

Members

"GetStartTime" method

Returns the start time of the "ISHCTimeSlice" instance.

```
CFRESULT GetStartTime(CFDATETIME64* pStartTime)
```

- `pStartTime`
  [out]: Time stamp with the start time of the time slice

"GetDuration" method

Supplies the duration of the "ISHCTimeSlice" instance.

```
CFRESULT GetDuration(CFTIMESPAN64* pDuration)
```

- `pDuration`
  [out]: The duration of the time slice

"GetCategory" method

Returns the time category time of the "ISHCTimeSlice" instance.

```
CFRESULT GetCategory(CFSTR* pstrCategoryName)
```

- `pstrCategoryName`
  [out]: The name of the time category

"SetCategory" method

Sets the time category of the "ISHCTimeSlice" instance.

```
CFRESULT SetCategory(CFSTR pstrCategoryName)
```

- `pstrCategoryName`
  [in]: The name of the new time category.

"SetStartTime" method

Sets the start time of the "ISHCTimeSlice" instance.

```
CFRESULT SetStartTime(CFDATETIME64 startTime)
```

- `startTime`
  [in]: The new start time of the time slice.

"SetDuration" method

Sets the duration of the "ISHCTimeSlice" instance.

```
CRFESULT SetDuration(CFTIMESPAN64 duration)
```

- `duration`
  [in]: The new duration of the time slice

ISHCTimeSliceEnumerator

Description

The C++ interface "ISHCTimeSliceEnumerator" specifies methods for handling the enumeration of the time slices of an "ISHCShiftTemplate" instance or "ISHCShift" instance.

The enumeration is returned by the `Read` method of these instances.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCTimeSlice** ppItem)
```

- `ppItem`
  [out]: The current "ISHCTimeSlice" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of time slices

Example

Copy code

```
void printTimeSlice(const ISHCTimeSliceEnumeratorPtr& p_pShcTimeSliceEnum)
{
   uint32_t nCout = 0;
   p_pShcTimeSliceEnum->Count(&nCout);
   for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
   {
      p_pShcTimeSliceEnum->MoveNext();
      cout << endl;
      ISHCTimeSlicePtr pTimeSlice;
      p_pShcTimeSliceEnum->Current(&pTimeSlice);
      CCfString strCategoryName;
      pTimeSlice->GetCategory(&strCategoryName);
      std::cout << "Category= " << strCategoryName.ToUTF8().c_str() << std::endl;
      CCfTimeSpan64 tsDuration;
      pTimeSlice->GetDuration(&tsDuration);
      CCfString strDuration = tsDuration.GetTimeSpanString();
      std::cout << "Durations= " << strDuration.ToUTF8().c_str() << std::endl;
      CCfDateTime64 dtStartTime;
      pTimeSlice->GetStartTime(&dtStartTime);
      CCfString strStarttime = dtStartTime.GetDateTimeString(false);
      std::cout << "Start Time= " << strStarttime.ToUTF8().c_str() << std::endl;
   }
}
```

ISHCDay

Description

The C++ interface "ISHCDay" specifies the methods of a day.

The interface inherits from the "ICfUnknown" interface.

Members

"CreateShift" method

Returns a new "ISHCShift" instance.

```
CRFESULT CreateShift(ISHCShiftTemplate* pShiftTemplate,
CFTIMESPAN64 startTime, ISHCShift** pShift)
```

- `pShiftTemplate`
  [in]: The "ISHCShiftTemplate" instance on which the new shift is to be based.
- `startTime`
  [in]: Time stamp for the start time of the new shift.
- `pShift`
  [out]: The new shift

"DeleteShift" method

Deletes an "ISHCShift" instance.

```
CRFESULT DeleteShift(ISHCShift* pShift)
```

- `pShift`
  [in]: Reference to the shift to be deleted.

"GetShifts" method

Supplies an "ISHCShiftEnumerator" instance via which you access the "ISHCShift" instances of the "ISHCDay" instance.

```
CRFESULT GetShifts(ISHCShiftEnumerator** ppSHCShiftEnumerator)
```

- `ppSHCShiftEnumerator`
  [out]: The enumerator with which you access the shifts of the "ISHCDay" instance. The shifts are contained in an array.

"GetComments" method

Supplies a map with the comments of the "ISHCDay" instance and their language code IDs.

```
CRFESULT GetComments(ICfMapIDToVariant** ppComments)
```

- `ppComments`
  [out]: A map with int32/string pairs (language code ID for comment).

Example:

```
ICfMapIDToVariantPtr pComments;
pShcshift->GetComments(&pComments);
uint32_t nCount = 0;
pComments->Count(&nCount);
for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
{
   int32_t nLanguageID;
   pComments->KeyAt(nIndex1, &nLanguageID);
   CCfVariant strComments;
   pComments->ValueAt(nLanguageID, &strComments);
   std::cout << "LangauageID =" << nLanguageID << " Comments=" <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
```

```
}
```

"SetComments" method

Adds a new comment in the specified language to the "ISHCDay" instance.

```
CRFESULT SetComment(CFLCID languageId , CFSTR pComments)
```

- `languageId`
  [in]: The language code ID
- `pComments`
  [in]: The comment text

"GetStartTime" method

Returns the start time of the "ISHCDay" instance.

```
CRFESULT GetStartTime(CFDATETIME64* pStartTime)
```

- `pStartTime`
  [out]: The start time

"SetStartTime" method

Sets the start time of the "ISHCDay" instance.

```
CRFESULT SetStartTime(CFDATETIME64 startTime)
```

- `startTime`
  [in]: The new start time

"GetIsCustomized" method

Supplies the information on whether the "ISHCDay" instance was edited by users.

```
CRFESULT GetIsCustomized)(CFBOOL* pIsCustomized)
```

- `pIsCustomized`
  [out]:
  - 0: Was not processed
  - 1: Was processed

"GetDayTemplate" method

Supplies the "ISHCDayTemplate" instance on which the "ISHCDay" instance is based.

```
CRFESULT GetDayTemplate(CFSTR* pDayTemplate)
```

- `pDayTemplate`
  [out]: The day template

"SetDayTemplate" method

Sets the "ISHCDayTemplate" instance of the "ISHCDay" instance.

```
CRFESULT SetDayTemplate(CFSTR pDayTemplate)
```

- `pDayTemplate`
  [in]: The new day template

See also

Locale IDs of the supported languages (Page 7829)

ISHCDayEnumerator

Description

The C++ interface "ISHCDayEnumerator" specifies methods for handling the enumeration of the days of an "ISHCCalendar" instance. The enumeration is returned by the "Read" method of an "ISHCDayProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCTimeSlice** ppItem)
```

- `ppItem`
  [out]: The current "ISHCDay" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of days

Example

Copy code

```
void PrintDay(const ISHCDayEnumeratorPtr& p_pShcdayEnum)
{
   uint32_t nCount = 0;
   p_pShcdayEnum->Count(&nCount);
   for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
   {
      cout << endl;
      p_pShcdayEnum->MoveNext();
      ISHCDayPtr pShcday;
      p_pShcdayEnum->Current(&pShcday);
      CCfString strDaytemplate;
      pShcday->GetDayTemplate(&strDaytemplate);
      cout << "DayTemplate= " << strDaytemplate.ToUTF8().c_str() << endl;

      ICfMapIDToVariantPtr pComments;
      pShcday->GetComments(&pComments);
      std::cout << "comments:-" << std::endl << std::endl;
      uint32_t nCount1 = 0;
      pComments->Count(&nCount1);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
      {
         int32_t nLanguageID;
         pComments->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strComments;
         pComments->ValueAt(nLanguageID, &strComments);
         std::cout << "LangauageID = " << nLanguageID << " Comments= " <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
      }
      CFBOOL bIsCustomized;
      pShcday->GetIsCustomized(&bIsCostomized);
      cout << "IsCustomized=" << (uint32_t)bIsCostomized << endl;
      CCfDateTime64 dtStartTime;
      pShcday->GetStartTime(&dtStartTime);
      cout << "StartTime= " << dtStartTime.GetDateTimeString(false).ToUTF8().c_str() <<
endl;

      ISHCShiftEnumeratorPtr pDayShifts;
      pShcday->GetShifts(&pDayShifts);
      //printShift(pDayShifts);
      uint32_t nCout = 0;
      pDayShifts->Count(&nCout);
      for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
      {
         cout << endl;
         pDayShifts->MoveNext();
         ISHCShiftPtr pShcshift;
         pDayShifts->Current(&pShcshift);
         CCfString strshiftTemplateName;
         pShcshift->GetShiftTemplate(&strshiftTemplateName);
         std::cout << "ShiftTemplateName= " << strshiftTemplateName.ToUTF8().c_str() <<
std::endl;
         CCfTimeSpan64 tsDuration;
         pShcshift->GetDuration(&tsDuration);
         CCfString strDuration = tsDuration.GetTimeSpanString();
```

Copy code

```
         std::cout << "Durations= " << strDuration.ToUTF8().c_str() << std::endl;
         CFBOOL bIsCustomised;
         pShcshift->GetIsCustomized(&bIsCustomised);
         std::cout << "Is Customised= " << (uint32_t)bIsCustomised << std::endl;
         CFENUM ndeltaKind;
         pShcshift->GetDeltaKind(&ndeltaKind);
         std::cout << "deltaKind = " << ndeltaKind << std::endl;
         uint32_t nShiftId;
         pShcshift->GetShiftId(&nShiftId);
         std::cout << "ShiftId = " << nShiftId << std::endl;
         ICfMapIDToVariantPtr pComments;
         pShcshift->GetComments(&pComments);
         std::cout << "comments:-" << std::endl << std::endl;
         uint32_t nCount = 0;
         pComments->Count(&nCount);
         for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
         {
            int32_t nLanguageID;
            pComments->KeyAt(nIndex1, &nLanguageID);
            CCfVariant strComments;
            pComments->ValueAt(nLanguageID, &strComments);
            std::cout << "LangauageID =" << nLanguageID << " Comments=" <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
         }
         ISHCTimeSliceEnumeratorPtr pShiftTimesliceEnum;
         pShcshift->GetTimeSlices(&pShiftTimesliceEnum);

         uint32_t nCout = 0;
         pShiftTimesliceEnum->Count(&nCout);
         for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
         {
            pShiftTimesliceEnum->MoveNext();
            cout << endl;
            ISHCTimeSlicePtr pTimeSlice;
            pShiftTimesliceEnum->Current(&pTimeSlice);
            CCfString strCategoryName;
            pTimeSlice->GetCategory(&strCategoryName);
            std::cout << "Category= " << strCategoryName.ToUTF8().c_str() << std::endl;

            CCfTimeSpan64 tsDuration;
            pTimeSlice->GetDuration(&tsDuration);
            CCfString strDuration = tsDuration.GetTimeSpanString();
            std::cout << "Durations= " << strDuration.ToUTF8().c_str() << std::endl;
            CCfDateTime64 dtStartTime;
            pTimeSlice->GetStartTime(&dtStartTime);
            CCfString strStarttime = dtStartTime.GetDateTimeString(false);
            std::cout << "Start Time= " << strStarttime.ToUTF8().c_str() << std::endl;
         }
      }
   }
}
```

ISHCDayProvider

Description

The C++ interface "ISHCDayProvider" provides you with access to an "ISHCDayEnumerator" instance which contains an enumeration with the days of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete days. The provider is returned by the "GetDayProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCDayEnumerator" instance which has access to an enumeration with the "ISHCDay" instances of a specific time period of the calendar.

```
CRFESULT Browse(CFDATETIME64 StartTime, CFDATETIME64 endTime,
ISHCDayEnumerator** ppISHCDayEnumerator)
```

- `startTime`
  [in]: Start time
- `endTime`
  [in]: End time
- `ppISHCDayEnumerator`
  [out]: The enumerator

Example:

Copy code

```
CCfTimeSpan64 Get1Hour()
{
     CCfDateTime64 dt = CCfDateTime64::Now(false);
     CFDATETIMEST st2;
     dt.GetDateTimeStruct(st2);
     st2.cHours = st2.cHours - 1;
     CCfDateTime64 dt2;
     dt2.SetFromDateTimeStruct(&st2);
     return dt.GetDifference(dt2);
}
CCfDateTime64 GetStartoftheDay()
{
     CCfDateTime64 dt = CCfDateTime64::Now(false);
     CFDATETIMEST st;
     dt.GetDateTimeStruct(st);
     st.cHours = 0;
     st.cMinutes = 0;
     st.cSeconds = 0;
     st.sHundredNanoSeconds = 0;
     st.sMicroSeconds = 0;
     st.sMilliSeconds = 0;
     dt.SetFromDateTimeStruct(&st);
     return dt;
}
ISHCDayEnumeratorPtr pDayEnum; // Get day instances for a timespan of three days
CFRESULT hr = pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24),
GetStartoftheDay() + (Get1Hour() * 48), &pDayEnum);
```

"Create" method

Adds new days to the enumeration with the "ISHCDay" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pDays)i
```

- `pDays`
  [in]: An array with the days to be added.

Example:

Copy code

```
void CreateDayWithShift()
{
   ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
   CCfArrayIUnknown arrDays;
   ICfArrayIUnknownPtr pDays;
   m_pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
   uint32_t nCount = 0;
   pShcDayTemplates->Count(&nCount);
   pShcDayTemplates->MoveNext();
   ISHCDayTemplatePtr pShcDayTemplate;
   pShcDayTemplates->Current(&pShcDayTemplate);
   CCfString strDayTemplateName;
   pShcDayTemplate->GetName(&strDayTemplateName);
   ICfUnknownPtr pUnk;
   m_pCalendar->GetObject(ODK_SHC_DAY, &pUnk);
   ISHCDayPtr pShcDay = (ISHCDayPtr)pUnk;
   pShcDay->SetDayTemplate(strDayTemplateName);
   CCfDateTime64 dt = GetStartoftheDay();
   pShcDay->SetStartTime(dt);
   pShcDay->SetComment(1033, CCfString(L"DayComments"));
   arrDays.Append(pShcDay);
   arrDays.DetachEnumerator(&pDays);
   m_pShcDayProvider->Create(pDays);
   ISHCShiftTemplateEnumeratorPtr pShcShiftTemplateEnum;
   m_pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplateEnum);
   pShcShiftTemplateEnum->MoveNext();
   ISHCShiftTemplatePtr pshcShiftTemplate;
   pShcShiftTemplateEnum->Current(&pshcShiftTemaplate);
   ISHCShiftPtr pShcdayShift;
   CCfTimeSpan64 starttime = Get1Hour() * 10;
   pShcDay->CreateShift(pshcShiftTemaplate, starttime, &pShcdayShift);
}
```

"Update" method

Updates "ISHCDay" instances of the enumeration.

`CRFESULTUpdate(ICfArrayIUnknown* pDays)`

- `pDays`
  [in]: An array with the days to be updated.

Example:

Copy code

```
void UpdateDayWithShift()
{
   ISHCDayEnumeratorPtr pDayEnum;
   m_pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24), GetStartoftheDay() +
(Get1Hour() * 24), &pDayEnum);
   pDayEnum->MoveNext();
   CCfArrayIUnknown ArrayDays;
   ICfArrayIUnknownPtr pArrayDays;
   ISHCDayPtr pShcDay;
   pDayEnum->Current(&pShcDay);
   CCfDateTime64 dt;
   pShcDay->GetStartTime(&dt);
   dt.AddTimeSpan(Get1Hour());
   pShcDay->SetStartTime(dt);//Update StartTime Not Supported
   pShcDay->SetComment(1033, CCfString(L"DayComment"));
   ISHCShiftEnumeratorPtr pShiftEnum;
   pShcDay->GetShifts(&pShiftEnum);
   pShiftEnum->MoveNext();
   ISHCShiftPtr pShcShift;
   pShiftEnum->Current(&pShcShift);
   CCfTimeSpan64 ts;
   pShcShift->GetDuration(&ts);
   pShcShift->SetDuration(ts + Get1Hour());
   pShcShift->SetComment(1033, CCfString(L"UpdatedShiftComments"));
   ArrayDays.Append(pShcDay);
   ArrayDays.DetachEnumerator(&pArrayDays);
   m_pShcDayProvider->Update(pArrayDays);
}
```

"Delete" method

Deletes "ISHCDay" instances of the calendar from the enumeration.

```
CRFESULT Delete(ICfArrayIUnknown* pActionTemplates)
```

- `pActionTemplates`
  [in]: An array with the days to be deleted.

Example:

Copy code

```
void DeleteDayWithShift()
{
   ISHCDayEnumeratorPtr pDayEnum;
   m_pShcDayProvider->Browse(GetStartoftheDay() - (Get1Hour() * 24), GetStartoftheDay() +
(Get1Hour() * 48), &pDayEnum);
   uint32_t nSize = 0;
   pDayEnum->Count(&nSize);
   CCfArrayIUnknown ArrayDays;
   ICfArrayIUnknownPtr pArrayDays;
   for (uint32_t nIdnex = 0;nIdnex < nSize;nIdnex++)
   {
      pDayEnum->MoveNext();
      ISHCDayPtr pShcDay;
      pDayEnum->Current(&pShcDay);
      ISHCShiftEnumeratorPtr pShiftEnum;
      pShcDay->GetShifts(&pShiftEnum);
      pShiftEnum->MoveNext();
      ISHCShiftPtr pShcShift;
      pShiftEnum->Current(&pShcShift);
      pShcDay->DeleteShift(pShcShift);
      ArrayDays.Append(pShcDay);
   }
   ArrayDays.DetachEnumerator(&pArrayDays);
   m_pShcDayProvider->Delete(pArrayDays);
}
```

ISHCDayTemplate

Description

The C++ interface "ISHCDayTemplate" specifies the methods of a day template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCDayTemplate" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- `pvarRet`
  [out]: The name of the day template

"SetName" method

Sets the name of the "ISHCDayTemplate" instance.

```
CRFESULT SetName(CFSTR  value)
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCDayTemplate" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for display name).

Example:

```
ICfMapIDToVariantPtr pDisplayNames;
pShcDayTemplate->GetDisplayNames(&pDisplayNames);
std::cout << "DisplayNames::" << std::endl << std::endl;
uint32_t nCount2 = 0;
pDisplayNames->Count(&nCount2);
for (uint32_t nIndex2 = 0; nIndex2 < nCount2; nIndex2++)
{
   int32_t nLanguageID;
   pDisplayNames->KeyAt(nIndex2, &nLanguageID);
   CCfVariant strDIsplayname;
   pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
   std::cout << "LangauageID =" << nLanguageID << " DisplayName="
<< CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
}
```

"SetDisplayName" method

Adds a new display name in the specified language to the "ISHCDayTemplate" instance.

```
CRFESULT SetDisplayName(CFLCID languageId, CFSTR pDisplayName)
```

- `languageId`
  [in]: The language code ID
- `pDisplayName`
  [in]: The comment text

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCDayTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for description).

Example: Similar to "GetDescriptions" of "ISHCCategory".

"SetDescription" method

Adds a new description in the specified language to the "ISHCDayTemplate" instance.

```
CRFESULT SetDescription)(CFLCID languageId, CFSTR pDescriptions)
```

- `languageId`
  [in]: The language code ID
- `pDescriptions`
  [in]: The description text

"GetShifts" method

Supplies an "ISHCShiftEnumerator" instance. The enumerator provides you with access to the "ISHCShift" instances of the "ISHCDayTemplate" instance.

```
GetShifts(ISHCShiftEnumerator** ppSHCShiftEnumerator)
```

- `ppSHCShiftEnumerator`
  [out]: The enumerator

"CreateShift" method

Adds an "ISHCShift" instance to the "ISHCDayTemplate" instance.

```
CRFESULT CreateShift(ISHCShiftTemplate* pShiftTemplate,
CFTIMESPAN64 pStartTime, ISHCShift** pShift)
```

- `pShiftTemplate`
  [in]: Reference to the shift template on which the shift is based.
- `pStartTime`
  [in] Time stamp with the start time of the shift
- `pShift`
  [out] reference to the new shift

"DeleteShift" method

Deletes an "ISHCShift" instance of the "ISHCDayTemplate" instance.

```
CFRESULTDeleteShift(ISHCShift* pShift)
```

- `pShift`
  [in]: Reference to the shift to be deleted

"GetIsDeleted" method

Supplies the information on whether the "ISHCDayTemplate" instance was deleted by users.
`CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)`

- `pIsDeleted`
  [out]:
  - 0: Was not deleted
  - 1: Was deleted

ISHCDayTemplatesProvider

Description

The C++ interface "ISHCDayTemplatesProvider" provides you with access to an "ISHCDayTemplateEnumerator" instance which contains an enumeration with the day templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete day templates. The provider is returned by the "GetDayTemplateProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCDayTemplateEnumerator" instance which has access to an enumeration with the "ISHCDayTemplate" instances of the calendar.

```
CRFESULT Browse(CFBOOL includeDeleted, ISHCDayTemplateEnumerator**
ppISHCDayTemplateEnumerator)
```

- `includeDeleted`
  Saves whether the enumerator also has access to the deleted day templates.
- `ppISHCDayTemplateEnumerator`
  [out]: The enumerator

Example:

```
ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
```

"Create" method

Adds new "ISHCDayTemplate" instances to the enumeration with the"ISHCDayTemplate" instances of the calendar.

```
CRFESULTUpdate(ICfArrayIUnknown* pDayTemplates)
```

- `pDayTemplates`
  [in]: An array with the day templates to be added.

Example:

Copy code

```
void CreateDayTemplateWithShift()
{
   ICfUnknownPtr pUnk;
   m_pCalendar->GetObject(ODK_SHC_DAY_TEMPLATE, &pUnk);
   CCfArrayIUnknown ArrayTemplate;
   ICfArrayIUnknownPtr pArrayTemplate;
   ISHCDayTemplatePtr pShcDayTemplate = (ISHCDayTemplatePtr)pUnk;
   pShcDayTemplate->SetName(CCfString(L"DaytemplateName"));
   pShcDayTemplate->SetDisplayName(1033, CCfString(L"DayTemplateDisplayName"));
   pShcDayTemplate->SetDescription(1033, CCfString(L"DayTemplate Descriptions"));

   ArrayTemplate.Append(pShcDayTemplate);
   ArrayTemplate.DetachEnumerator(&pArrayTemplate);
   m_pShcDayTemplateProvider->Create(pArrayTemplate);
   ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
   m_pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
   pShcShiftTemplates->MoveNext();
   ISHCShiftTemplatePtr pShcshiftTemplate;
   pShcShiftTemplates->Current(&pShcshiftTemplate);
   CCfString strShiftTemplateName;
   pShcshiftTemplate->GetName(&strShiftTemplateName);
   ISHCShiftPtr pShift;
   pShcDayTemplate->CreateShift(pShcshiftTemplate, Get1Hour() * 1, &pShift);
}
```

"Update" method

Updates "ISHCDayTemplate" instances of the enumeration.

```
CRFESULTUpdate(ICfArrayIUnknown* pDayTemplates)
```

- `pDayTemplates`
  [in]: An array with the day templates to be updated.

Example:

Copy code

```
void UpdateDayTemplateWithShift()
{
   ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
   hr = m_pShcDayTemplateProvider->Browse(false, &pShcDayTemplates);
   CCfArrayIUnknown ArrayDayTemplate;
   ICfArrayIUnknownPtr pArrayDayTemplate;
   uint32_t nCount = 0;
   pShcDayTemplates->Count(&nCount);
   for (uint32_t i = 0; i < nCount; i++)
   {
      pShcDayTemplates->MoveNext();
      ISHCDayTemplatePtr pShcDayTemplate;
      pShcDayTemplates->Current(&pShcDayTemplate);
      pShcDayTemplate->SetName(CCfString(L"UpdatedDayTemplateName"));
      ISHCShiftEnumeratorPtr pShiftEnum;
      pShcDayTemplate->GetShifts(&pShiftEnum);
      uint32_t nlength = 0;
      pShiftEnum->Count(&nlength);
      for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
      {
         pShiftEnum->MoveNext();
         ISHCShiftPtr pShcShift;
         pShiftEnum->Current(&pShcShift);
         ISHCTimeSliceEnumeratorPtr pShcTimeSliceEnum;
         pShcShift->GetTimeSlices(&pShcTimeSliceEnum);
         pShcShift->SetDuration(Get1Hour() * 8);
         pShcShift->SetComment(1033, CCfString("ShiftComment"));
         uint32_t nCount1 = 0;
         pShcTimeSliceEnum->Count(&nCount1);
         for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
         {
            pShcTimeSliceEnum->MoveNext();
            ISHCTimeSlicePtr pShctimeslice;
            pShcTimeSliceEnum->Current(&pShctimeslice);
            pShctimeslice->SetCategory(CCfString(L"Working"));
         }
         ArrayDayTemplate.Append(pShcDayTemplate);
   }
   ArrayDayTemplate.DetachEnumerator(&pArrayDayTemplate);
  m_pShcDayTemplateProvider->Update(pArrayDayTemplate);
}
```

"Delete" method

Deletes "ISHCDayTemplate" instances of the calendar from the enumeration.

```
CRFESULT Delete(ICfArrayIUnknown* pDayTemplates)
```

- `pDayTemplates`
  [in]: An array with the day templates to be deleted.

Example:

Copy code

```
void DeleteDaytemplateWithShift()
{
   ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
   m_pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
   CCfArrayIUnknown ArrayDayTemplate;
   ICfArrayIUnknownPtr pArrayDayTemplate;
   uint32_t nCount = 0;
   pShcDayTemplates->Count(&nCount);
   for (uint32_t i = 0; i < nCount; i++)
   {
      pShcDayTemplates->MoveNext();
      ISHCDayTemplatePtr pShcDayTemplate;
      pShcDayTemplates->Current(&pShcDayTemplate);
      ISHCShiftEnumeratorPtr pShiftEnum;
      pShcDayTemplate->GetShifts(&pShiftEnum);
      uint32_t nlength = 0;
      pShiftEnum->Count(&nlength);
      for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
      {
         pShiftEnum->MoveNext();
         ISHCShiftPtr pShcShift;
         pShiftEnum->Current(&pShcShift);
         pShcDayTemplate->DeleteShift(pShcShift);
      }
      ArrayDayTemplate.Append(pShcDayTemplate);

      ArrayDayTemplate.DetachEnumerator(&pArrayDayTemplate);
      m_pShcDayTemplateProvider->Delete(pArrayDayTemplate);
   }
}
```

ISHCShiftTemplate

Description

The C++ interface "ISHCShiftTemplate" specifies the methods of a shift template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCShiftTemplate" instance.

```
GetName(CFSTR * pvarRet)
```

- `pvarRet`
  [out]: Name

"SetName" method

Sets the name of the "ISHCShiftTemplate" instance.

```
CRFESULT SetName(CFSTR  value)
```

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
CFRESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for display name).

"SetDisplayName" method

Adds a new entry to a map with the display names of the "ISHCShiftTemplate" instance and their language code IDs.

```
CRFESULT SetDisplayName(CFLCID languageId, CFSTR pDisplayName)
```

- `languageId`
  [in]: The language code ID
- `pDisplayName`
  [in]: The comment text

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCShiftTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: A map with int32/string pairs (language code ID for description).

"SetDescription" method

Adds a new entry to a map with the description of the "ISHCShiftTemplate" instance and their language code IDs.

```
CRFESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- `languageId`
  [in]: The language code ID
- `pDescriptions`
  [in]: The description text

"GetIsDeleted" method

Supplies the information on whether the "ISHCShiftTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- `pIsDeleted`
  [out]:
  - 0: Was not deleted
  - 1: Was deleted

"GetDuration" method

Supplies the duration of the "ISHCShiftTemplate" instance.

```
CFRESULT GetDuration(CFTIMESPAN64* pDuration)
```

- `pDuration`
  [out]: The duration of the shift template

"SetDuration" method

Sets the duration of the "ISHCShiftTemplate" instance.

```
CFRESULT SetDuration(CFTIMESPAN64 duration)
```

- `duration`
  [in]: The duration of the shift template

"GetTimeSlices" method

Supplies an "ISHCTimeSliceEnumerator" instance. The enumerator provides you with access to the "ISHCTimeSlice" instances of the "ISHCShiftTemplate" instance.

```
CRFESULT GetTimeSlices(ISHCTimeSliceEnumerator**
ppSHCTimeSliceEnumerator)
```

- `ppSHCTimeSliceEnumerator`
  [out]: The enumerator

"CreateTimeSlice" method

Adds an "ISHCTimeSlice" instance to the "ISHCShiftTemplate" instance.

```
CRFESULT CreateTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the new time slice

"DeleteTimeSlice" method

Deletes an "ISHCTimeSlice" instance of the "ISHCShiftTemplate" instance.

```
CRFESULT DeleteTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the time slice to be deleted

ISHCShiftTemplateEnumerator

Description

The C++ interface "ISHCShiftTemplateEnumerator" specifies methods for handling the enumeration of the shift templates of an "ISHCCalendar" instance. The enumeration is returned by the "Read" method of an "ISHCShiftTemplatesProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCShiftTemplate** ppItem)
```

- `ppItem`
  [out]: The current shift template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

Supplies the number of shift templates in the list.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of shift templates

Example

Copy code

```
void PrintShiftTemplate(const ISHCShiftTemplateEnumeratorPtr& p_pShcShiftTemplateEnum)
{
   uint32_t nCout = 0;
   p_pShcShiftTemplateEnum->Count(&nCout);
   for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
   {
      cout << endl;
      p_pShcShiftTemplateEnum->MoveNext();
      ISHCShiftTemplatePtr pShcShiftTemplate;
      p_pShcShiftTemplateEnum->Current(&pShcShiftTemplate);
      CCfString strName;
      pShcShiftTemplate->GetName(&strName);
      std::cout << "Name=" << strName.ToUTF8().c_str() << std::endl;

      CCfTimeSpan64 tsDuration;
      pShcShiftTemplate->GetDuration(&tsdurantion);
      CCfString strDuration = tsdurantion.GetTimeSpanString();
      std::cout << "Duration=" << strDuration.ToUTF8().c_str() << std::endl;

      CFBOOL bIsDeleted;
      pShcShiftTemplate->GetIsDeleted(&bIsDeleted);
      std::cout << "ISDeleted=" << (uint32_t)bIsDeleted << std::endl;

      ICfMapIDToVariantPtr pDescriptions;
      pShcShiftTemplate->GetDescriptions(&pDescriptions);
      std::cout << "Descriptions=" << std::endl << std::endl;
      uint32_t nCount = 0;
      pDescriptions->Count(&nCount);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
      {
         int32_t nLanguageID;
         pDescriptions->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strDescription;
         pDescriptions->ValueAt(nLanguageID, &strDescription);
         std::cout << "LangauageID =" << nLanguageID << " Description=" <<
CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
      }

      ICfMapIDToVariantPtr pDisplayNames;
      pShcShiftTemplate->GetDisplayNames(&pDisplayNames);
      std::cout << "DisplayNames::" << std::endl << std::endl;
      uint32_t nCount = 0;
      pDisplayNames->Count(&nCount);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
      {
         int32_t nLanguageID;
         pDisplayNames->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strDIsplayname;
         pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
         std::cout << "LangauageID =" << nLanguageID << " DisplayName=" <<
CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
      }
      ISHCTimeSliceEnumeratorPtr pShiftTemplatetimeSlice;
      pShcShiftTemplate->GetTimeSlices(&pShiftTemplatetimeSlice);
```

Copy code

```
        uint32_t nCout = 0;
        pShiftTemplatetimeSlice->Count(&nCout);
        for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
        {
           pShiftTemplatetimeSlice->MoveNext();
           cout << endl;
           ISHCTimeSlicePtr pTimeSlice;
           pShiftTemplatetimeSlice->Current(&pTimeSlice);
           CCfString strCategoryName;
           pTimeSlice->GetCategory(&strCategoryName);
           std::cout << "Category= " << strCategoryName.ToUTF8().c_str() << std::endl;

           CCfTimeSpan64 tsDuration;
           pTimeSlice->GetDuration(&tsDuration);
           CCfString strDuration = tsDuration.GetTimeSpanString();
           std::cout << "Durations= " << strDuration.ToUTF8().c_str() << std::endl;
           CCfDateTime64 dtStartTime;
           pTimeSlice->GetStartTime(&dtStartTime);
           CCfString strStarttime = dtStartTime.GetDateTimeString(false);
           std::cout << "Start Time= " << strStarttime.ToUTF8().c_str() << std::endl;
        }
     }
}
```

ISHCShiftTemplatesProvider

Description

The C++ interface "ISHCShiftTemplatesProvider" provides you with access to an "ISHCShiftTemplateEnumerator" instance which contains an enumeration with the shift templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete shift templates. The provider is returned by the "GetShiftTemplateProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCShiftTemplateEnumerator" instance which has access to an enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CRFESULT Browse(CFBOOL includeDeleted,
ISHCShiftTemplateEnumerator** ppISHCShiftTemplateEnumerator)
```

- `includeDeleted`
  Saves whether the enumerator also has access to the deleted shift templates.
- `ppISHCShiftTemplateEnumerator`
  [out]: The enumerator

Example:

```
ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
CFRESULT hr = pShcDayTemplateProvider->Browse(CF_FALSE,
&pShcDayTemplates);
```

"Create" method

Adds new shift templates to the enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pShiftTemplates)
```

- `pShiftTemplates`
  [in]: An array with the shift templates to be added.

Example:

Copy code

```
void CreateShiftTemplateWithTimeslice()
{
   ISHCShiftTemplatePtr pShcShiftTemplate;
   ICfUnknownPtr pUnk;
   m_pCalendar->GetObject(ODK_SHC_SHIFT_TEMPLATE, &pUnk);
   pShcShiftTemplate = (ISHCShiftTemplatePtr)pUnk;
   pShcShiftTemplate->SetName(CCfString(L"ShiftTemplateName"));
   pShcShiftTemplate->SetDisplayName(1033, CCfString(L"ShiftDisplayName"));
   pShcShiftTemplate->SetDescription(1033, CCfString(L"ShiftTemplateDescription"));
   pShcShiftTemplate->SetDuration(Get1Hour() * 8);

   ICfArrayIUnknownPtr pArrayShiftTemplate;
   CCfArrayIUnknown ArrayShiftTemplate;
   ArrayShiftTemplate.Append(pShcShiftTemplate);
   ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
   m_pShcShiftTemplateProvider->Create(pArrayShiftTemplate);

   ICfUnknownPtr pUnkTimeSlice;
   m_pCalendar->GetObject(ODK_SHC_TIME_SLICE, &pUnkTimeSlice);
   ISHCTimeSlicePtr pShcTimeSlice;
   pUnkTimeSlice->QueryInterface(IID_ISHCTimeSlice, (ICfUnknown**)&pShcTimeSlice);
   pShcTimeSlice->SetCategory(CCfString(L"Working"));
   pShcTimeSlice->SetDuration(Get1Hour() * 1);
   CCfDateTime64 dt = GetStartoftheDay();
   pShcTimeSlice->SetStartTime(dt);
   pShcShiftTemplate->CreateTimeSlice(pShcTimeSlice);
}
```

"Update" method

Updates the enumeration with the "ISHCShiftTemplate" instances of the calendar.

```
CRFESULT Update(ICfArrayIUnknown* pShiftTemplates)
```

- `pShiftTemplates`
  [in]: An array with the shift templates to be updated.

Example:

Copy code

```
void UpdateShiftTemplateWithTimeSlice()
{
   ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
   m_pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
   CCfArrayIUnknown ArrayShiftTemplate;
   ICfArrayIUnknownPtr pArrayShiftTemplate;
   uint32_t nCount = 0;
   pShcShiftTemplates->Count(&nCount);
   for (uint32_t i = 0; i < nCount; i++)
   {
      pShcShiftTemplates->MoveNext();
      ISHCShiftTemplatePtr pShcShiftTemplate;
      pShcShiftTemplates->Current(&pShcShiftTemplate);
      pShcShiftTemplate->SetDuration(Get1Hour() * 6);
      pShcShiftTemplate->SetName(CCfString(L"UpdatedShiftTemplateName"));

      ISHCTimeSliceEnumeratorPtr pTimeSlilceEnum;
      pShcShiftTemplate->GetTimeSlices(&pTimeSlilceEnum);
      uint32_t nlength = 0;
      pTimeSlilceEnum->Count(&nlength);
      for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
      {
         pTimeSlilceEnum->MoveNext();
         ISHCTimeSlicePtr pShcTimeSlice;
         pTimeSlilceEnum->Current(&pShcTimeSlice);
         pShcTimeSlice->SetDuration(Get1Hour() * 4);
         pShcTimeSlice->SetCategory(CCfString(L"Break"));
      }
      ArrayShiftTemplate.Append(pShcShiftTemplate);
   }
   ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
   m_pShcShiftTemplateProvider->Update(pArrayShiftTemplate);
}
```

"Delete" method

Deletes an "ISHCShiftTemplate" instance of the calendar from the enumeration.

```
CRFESULT Delete(ICfArrayIUnknown* pShiftTemplates)
```

- `pShiftTemplates`
  [in]: An array with the shift templates to be deleted.

Example:

Copy code

```
void DeleteShiftTemplateWithTimeSlice()
{
   ISHCShiftTemplateEnumeratorPtr pShcShiftTemplates;
   m_pShcShiftTemplateProvider->Browse(CF_FALSE, &pShcShiftTemplates);
   CCfArrayIUnknown ArrayShiftTemplate;
   ICfArrayIUnknownPtr pArrayShiftTemplate;
   uint32_t nCount = 0;
   pShcShiftTemplates->Count(&nCount);
   for (uint32_t i = 0; i < nCount; i++)
   {
      pShcShiftTemplates->MoveNext();
      ISHCShiftTemplatePtr pShcShiftTemplate;
      pShcShiftTemplates->Current(&pShcShiftTemplate);
      ISHCTimeSliceEnumeratorPtr pTimeSlilceEnum;
      pShcShiftTemplate->GetTimeSlices(&pTimeSlilceEnum);
      uint32_t nlength = 0;
      pTimeSlilceEnum->Count(&nlength);
      for (uint32_t nIndex = 0; nIndex < nlenght; nIndex++)
      {
         pTimeSlilceEnum->MoveNext();
         ISHCTimeSlicePtr pShcTimeSlice;
         pTimeSlilceEnum->Current(&pShcTimeSlice);
         pShcShiftTemplate->DeleteTimeSlice(pShcTimeSlice);
      }
      ArrayShiftTemplate.Append(pShcShiftTemplate);
   }
   ArrayShiftTemplate.DetachEnumerator(&pArrayShiftTemplate);
   m_pShcShiftTemplateProvider->Delete(pArrayShiftTemplate);
}
```

ISHCShift

Description

The C++ interface "ISHCShift" specifies the methods of a shift.

The interface inherits from the "ICfUnknown" interface.

Members

"GetTimeSlices" method

Supplies an "ISHCTimeSliceEnumerator" instance. The enumerator provides you with access to the "ISHCTimeSlice" instances of the "ISHCShift" instance.

```
CRFESULT GetTimeSlices(ISHCTimeSliceEnumerator**
ppSHCTimeSliceEnumerator)
```

- `ppSHCTimeSliceEnumerator`
  [out]: The enumerator

"CreateTimeSlice" method

Adds an "ISHCTimeSlice" instance to the "ISHCShiftTemplate" instance.

```
CRFESULT CreateTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the new time slice

Example

Copy code

```
void CreateTimeSliceUsingShift()
{
   ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
   m_pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
   uint32_t nCount = 0;
   pShcDayTemplates->Count(&nCount);

   pShcDayTemplates->MoveNext();
   ISHCDayTemplatePtr pShcDayTemplate;
   pShcDayTemplates->Current(&pShcDayTemplate);
   ISHCShiftEnumeratorPtr pShiftEnum;
   pShcDayTemplate->GetShifts(&pShiftEnum);
   uint32_t nlength = 0;
   pShiftEnum->Count(&nlength);
   pShiftEnum->MoveNext();
   ISHCShiftPtr pShcShift;
   pShiftEnum->Current(&pShcShift);
   ISHCTimeSlicePtr pShcTimeSlice;
   ICfUnknownPtr pUnk;
   m_pCalendar->GetObject(ODK_SHC_TIME_SLICE, &pUnk);
   pShcTimeSlice = (ISHCTimeSlicePtr)pUnk;
   pShcTimeSlice->SetCategory(CCfString(L"Break"));
   pShcTimeSlice->SetDuration(Get1Hour() * 1);
   CCfDateTime64 dt = GetStartoftheDay();
   dt.AddTimeSpan(Get1Hour() * 3);
   pShcTimeSlice->SetStartTime(dt);

   pShcShift->CreateTimeSlice(pShcTimeSlice);
}
```

"DeleteTimeSlice" method

Deletes an "ISHCTimeSlice" instance of the "ISHCShiftTemplate" instance.

```
CRFESULT DeleteTimeSlice(ISHCTimeSlice* pSlice)
```

- `pSlice`
  [in]: Reference to the time slice to be deleted

Example:

Copy code

```
void DeleteTimeSliceUsingShift()
{
   ISHCDayTemplateEnumeratorPtr pShcDayTemplates;
   m_pShcDayTemplateProvider->Browse(CF_FALSE, &pShcDayTemplates);
   uint32_t nCount = 0;
   pShcDayTemplates->Count(&nCount);
   pShcDayTemplates->MoveNext();
   ISHCDayTemplatePtr pShcDayTemplate;
   ISHCShiftEnumeratorPtr pShiftEnum;
   pShcDayTemplate->GetShifts(&pShiftEnum);
   uint32_t nlength = 0;
   pShiftEnum->Count(&nlength);
   pShiftEnum->MoveNext();
   ISHCShiftPtr pShcShift;
   pShiftEnum->Current(&pShcShift);
   ISHCTimeSliceEnumeratorPtr pShcTimeSliceEnum;
   pShcShift->GetTimeSlices(&pShcTimeSliceEnum);
   uint32_t nCountTimeSlice = 0;
   pShcTimeSliceEnum->Count(&nCountTimeSlice);
   pShcTimeSliceEnum->MoveNext();
   ISHCTimeSlicePtr pShcTimeSlice;
   pShcTimeSliceEnum->Current(&pShcTimeSlice);
   pShcShift->DeleteTimeSlice(pShcTimeSlice);
}
```

"GetComments" method

Supplies a map with the comments of the "ISHCShift" instance and their language code IDs.

```
CRFESULT GetComments(ICfMapIDToVariant** ppComments)
```

- `ppComments`
  [out]: A map with int32/string pairs (language code ID for comment).

"SetComments" method

Adds a new entry to a map with the comments of the "ISHCShift" instance and their language code IDs.

```
CRFESULT SetComment(CFLCID languageId , CFSTR pComments)
```

- `languageId`
  [in]: The language code ID
- `pComments`
  [in]: The comment text

"GetDuration" method

Supplies the duration of the "ISHCShift" instance.

```
CFRESULT GetDuartion(CFTIMESPAN64* pDuration)
```

- `pDuration`
  [out]: The duration of the shift

"SetDuration" method

Sets the duration of the "ISHCShift" instance.

```
CFRESULT SetDuration(CFTIMESPAN64 duration)
```

- `duration`
  [in]: The duration of the shift

"GetShiftTemplate" method

Supplies the "ISHCShiftTemplate" instance on which the "ISHCShift" instance is based.

```
CRFESULT GetShiftTemplate(CFSTR* pSHCShiftTemplate)
```

- `pSHCShiftTemplate`
  [out]: The shift template

"CreateAction" method

Adds an "ISHCAction" instance to the "ISHCShift" instance.

```
CRFESULT CreateAction(ISHCActionTemplate* pActionTemplate,
CFTIMESPAN64 offset, ISHCAction** pShcAction)
```

- `pActionTemplate`
  [in]: The action template on which the new action is to be based.
- `offset`
  [in]: The offset of the action in relation to the start time of the "ISHCShift" instance. Positive and negative value allowed.
- `pShcAction`
  [out]: The new action

"DeleteAction" method

Deletes an "ISHCAction" instance of the "ISHCShift" instance.

```
CFRESULT DeleteAction(ISHCAction* pShcAction)
```

- `pShcAction`
  [in]: Reference to the action to be deleted.

"GetAction" method

Supplies an "ISHCActionEnumerator" instance via which you access the "ISHCAction" instances of the "ISHCShift" instance.

```
CRFESULT GetActions(ISHCActionEnumerator** ppSHCActionEnumerator)
```

- `ppSHCActionEnumerator`
  [out]: The enumerator

"GetIsCustomized" method

Supplies the information on whether the "ISHCShift" instance was edited by users.

```
CRFESULT GetIsCustomized)(CFBOOL* pIsCustomized)
```

- `pIsCustomized`
  [out]:
  - 0: Was not processed
  - 1: Was processed

"GetDeltaKind" method

Supplies the delta type of the "ISHCShift" instance.

```
CRFESULT GetDeltaKind(CFENUM* pDeltaKind)
```

- `pDeltaKind`
  [out]: Points to the enumeration "ShcDeltaType", which can contain the following values:
  - Added (0)
  - Modified (1)
  - Deleted (2)

"GetShiftId" method

Supplies the ID of the "ISHCShift" instance.

```
CFRESULT GetShiftId(uint32_t* pShiftId)
```

- `pShift`
  [out]: The ID

"SetShiftId" method

Sets the ID of the "ISHCShift" instance.

```
SetShiftId(uint32_t ShiftId)
```

- `ShiftId`
  [in]: The new ID

See also

Locale IDs of the supported languages (Page 7829)

ISHCShiftEnumerator

Description

The C++ interface "ISHCShiftEnumerator" specifies methods for handling the enumeration of shifts of an "ISHCDay" instance or "ISHCDayTemplate" instance.

The enumeration is returned by the "GetShifts" method of these instances.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCShift** ppItem)
```

- `ppItem`
  [out]: The current shift

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of shifts

Example

Copy code

```
void printShift(const ISHCShiftEnumeratorPtr& p_pShcShiftEnum)
{
   uint32_t nCout = 0;
   p_pShcShiftEnum->Count(&nCout);
   for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
   {
      cout << endl;
      p_pShcShiftEnum->MoveNext();
      ISHCShiftPtr pShcshift;
      p_pShcShiftEnum->Current(&pShcshift);
      CCfString strshiftTemplateName;
      pShcshift->GetShiftTemplate(&strshiftTemplateName);
      std::cout << "ShiftTemplateName= " << strshiftTemplateName.ToUTF8().c_str() <<
std::endl;
      CCfTimeSpan64 tsDuration;
      pShcshift->GetDuration(&tsDuration);
      CCfString strDuration = tsDuration.GetTimeSpanString();
      std::cout << "Durations= " << strDuration.ToUTF8().c_str() << std::endl;
      CFBOOL bIsCustomised;
      pShcshift->GetIsCustomized(&bIsCustomised);
      std::cout << "Is Customised= " << (uint32_t)bIsCustomised << std::endl;
      CFENUM ndeltaKind;
      pShcshift->GetDeltaKind(&ndeltaKind);
      std::cout << "deltaKind = " << ndeltaKind << std::endl;
      uint32_t nShiftId;
      pShcshift->GetShiftId(&nShiftId);
      std::cout << "ShiftId = " << nShiftId << std::endl;
      ICfMapIDToVariantPtr pComments;
      pShcshift->GetComments(&pComments);
      std::cout << "comments:-" << std::endl << std::endl;
      uint32_t nCount = 0;
      pComments->Count(&nCount);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount; nIndex1++)
      {
         int32_t nLanguageID;
         pComments->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strComments;
         pComments->ValueAt(nLanguageID, &strComments);
         std::cout << "LangauageID =" << nLanguageID << " Comments=" <<
CCfSmartString(strComments).ToUTF8().c_str() << std::endl;
      }
      ISHCTimeSliceEnumeratorPtr pShiftTimesliceEnum;
      pShcshift->GetTimeSlices(&pShiftTimesliceEnum);

      uint32_t nCout = 0;
      pShiftTimesliceEnum->Count(&nCout);
      for (uint32_t nIndex = 0; nIndex < nCout; nIndex++)
      {
         pShiftTimesliceEnum->MoveNext();
         cout << endl;
         ISHCTimeSlicePtr pTimeSlice;
         pShiftTimesliceEnum->Current(&pTimeSlice);
         CCfString strCategoryName;
         pTimeSlice->GetCategory(&strCategoryName);
```

Copy code

```
            std::cout << "Category= " << strCategoryName.ToUTF8().c_str() << std::endl;

            CCfTimeSpan64 tsDuration;
            pTimeSlice->GetDuration(&tsDuration);
            CCfString strDuration = tsDuration.GetTimeSpanString();
            std::cout << "Durations= " << strDuration.ToUTF8().c_str() << std::endl;
            CCfDateTime64 dtStartTime;
            pTimeSlice->GetStartTime(&dtStartTime);
            CCfString strStarttime = dtStartTime.GetDateTimeString(false);
            std::cout << "Start Time= " << strStarttime.ToUTF8().c_str() << std::endl;
        }
    }
}
```

ISHCAction

Description

The C++ interface "ISHCAction" specifies the methods of an action.

The interface inherits from the "ICfUnknown" interface.

Members

"GetOffset" method

Returns the offset of the "ISHCAction" instance in relation to the starting point of its "ISHCShift" instance.

```
CFRESULT GetOffset(CFTIMESPAN64* pOffsetType)
```

- `pOffset`
  [out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCAction" instance in relation to the starting point of its "ISHCShift" instance.

```
CFRESULT GetOffset(CFTIMESPAN64 OffsetType)
```

- `OffsetType`
  [in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetActionTemplate" method

Supplies the "ISHCActionTemplate" instance of the "ISHCAction" instance.

```
GetActionTemplate(CFSTR* p_strActionTemplate)
```

- `p_strActionTemplate`
  [out]: The action template

"GetIsCustomized" method

Supplies the information on whether the "ISHCAction" instance was edited by users.

```
CRFESULT GetIsCustomized(CFBOOL* pIsCustomized)
```

- `pIsCustomized`
  [out]:
  - 0: Was not processed
  - 1: Was processed

"GetElements" method

Supplies an "ISHCActionElementEnumerator" instance via which you access the action elements of the "ISHCAction" instance.

```
GetElements(ISHCActionElementEnumerator**
ppSHCActionElementEnumerator)
```

- `ppSHCActionElementEnumerator`
  [out]: The enumerator

ISHCActionEnumerator

Description

The C++ interface "ISHCActionEnumerator" specifies methods for handling the enumeration of actions of an "ISHCShift" instance or "ISHCShiftTemplate" instance.

The enumeration is returned by the "GetActions" method of these instances.

The interface inherits from the "IcfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCAction** ppItem)
```

- `ppItem`
  [out]: The current action

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of actions

Example

Copy code

```
void PrintAction(const ISHCActionEnumeratorPtr& pShcActionEnum)
{
   uint32_t nCount = 0;
   pShcActionEnum->Count(&nCount);
   for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
   {
      cout << endl;
      pShcActionEnum->MoveNext();
      ISHCActionPtr pShcAction;
      pShcActionEnum->Current(&pShcAction);
      CCfString strActionTemplateName;
      pShcAction->GetActionTemplate(&strActionTemplateName);
      std::cout << "ActionTemplateName = " << strActionTemplateName.ToUTF8().c_str() <<
std::endl;
      CFBOOL isCustomize;
      pShcAction->GetIsCustomized(&isCustomize);
      cout << "IsCustomize= " << (uint32_t)isCustomize << endl;

      CCfTimeSpan64 offset;
      hr = pShcAction->GetOffset(&offset);
      cout << "Offset=" << offset.GetTimeSpanString().ToUTF8().c_str() << endl;
      ISHCActionElementEnumeratorPtr pShcActionElementEnum;
      pShcAction->GetElements(&pShcActionElementEnum);

      cout << endl << "******PrintActionElement*******" << endl << endl;
      uint32_t nCount = 0;
      pShcActionElementEnum->Count(&nCount);
      for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
      {
         cout << endl;
         pShcActionElementEnum->MoveNext();
         ISHCActionElementPtr pShcActionElement;
         pShcActionElementEnum->Current(&pShcActionElement);
         CCfString strElementName;
         pShcActionElement->GetElementName(&strElementName);
         cout << "ActionElementName= " << strElementName.ToUTF8().c_str() << endl;
         CFENUM nType;
         pShcActionElement->GetElementType(&nType);
         cout << "Element Type= " << nType << endl;
         CFBOOL bIsEnable;
         pShcActionElement->GetEnabled(&bIsEnable);
         cout << "Is Enable = " << (uint32_t)bIsEnable << endl;
         CCfTimeSpan64 offset;
         pShcActionElement->GetOffset(&offset);
         cout << "Offset = " << offset.GetTimeSpanString().ToUTF8().c_str() << endl;
         CCfVariant vtValue;
         pShcActionElement->GetValue(&vtValue);
         cout << "value = " << vtValue.uint32 << endl;
      }
   }
}
```

ISHCActionElement

Description

The C++ interface "ISHCActionElement" specifies the methods of the action elements of an "ISHCAction" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"GetElementType" method

Supplies the type of the "ISHCActionElement" instance.

```
CRFESULT GetElementType(CFENUM* pElementType)
```

- `pElementType`
  [out]: Points to the enumeration "ShcActionElementType", which can contain the following values:
  - `Tag` (0)
    The action element controls a tag.

"GetEnabled" method

Supplies the information on whether the "ISHCActionElement" instance is activated.

```
CRFESULT GetEnabled(CFBOOL* pEnabled)
```

- `pEnabled`
  [out]:
  - 0: Deactivated
  - 1: Activated

"SetEnabled" method

Sets whether the "ISHCActionElement" instance is activated.

```
CRFESULT SetEnabled(CFBOOL Enabled)
```

- `Enabled`
  [in]:
  - 0: Deactivated
  - 1: Activated

"GetOffset" method

Returns the offset of the "ISHCActionElement" instance in relation to the anchor point of its "ISHCAction" instance.

```
CFRESULT GetOffset(CFTIMESPAN64* pOffset)
```

- `pOffset`
  [out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCActionElement" instance in relation to the anchor point of its "ISHCAction" instance.

```
CFRESULT SetOffset(CFTIMESPAN64 offset)
```

- `offset`
  [in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetValue" method

Supplies the value of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- `pValue`
  [out]: The tag value

"SetValue" method

Sets the value of the tag controlled by the "ISHCActionElement" instance.

```
CRFESULT SetValue(CFVARIANT value)
```

- `value`
  [in]: The new tag value

"GetElementName" method

Supplies the name of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT GetElementName(CFSTR* p_strActionElement)
```

- `p_strActionElement`
  [out]: The tag name

"SetElementName" method

Sets the name of the tag controlled by the "ISHCActionElement" instance.

```
CFRESULT SetElementName(CFSTR ActionElement)
```

- `ActionElement`
  [in]: The tag name

ISHCActionElementEnumerator

Description

The C++ interface "ISHCActionElementEnumerator" specifies methods for handling the enumeration of action elements of an "ISHCAction" instance.

The enumeration is returned by the "GetElements" method of an "ISHCAction" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionElement** ppItem)
```

- `ppItem`
  [out]: The current action element

"Reset" method

Reset the current position in enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of action elements

Example

Copy code

```
 void PrintActionElement(const ISHCActionElementEnumeratorPtr& pShcActionElementEnum)
{
      cout << endl << "******PrintActionElement*******" << endl << endl;
      uint32_t nCount = 0;
      pShcActionElementEnum->Count(&nCount);
      for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
      {
         cout << endl;
         pShcActionElementEnum->MoveNext();
         ISHCActionElementPtr pShcActionElement;
         pShcActionElementEnum->Current(&pShcActionElement);
         CCfString strElementName;
         pShcActionElement->GetElementName(&strElementName);
         cout << "ActionElementName= " << strElementName.ToUTF8().c_str() << endl;
         CFENUM nType;
         pShcActionElement->GetElementType(&nType);
         cout << "Element Type= " << nType << endl;
         CFBOOL bIsEnable;
         pShcActionElement->GetEnabled(&bIsEnable);
         cout << "Is Enable = " << (uint32_t)bIsEnable << endl;
         CCfTimeSpan64 offset;
         pShcActionElement->GetOffset(&offset);
         cout << "Offset = " << offset.GetTimeSpanString().ToUTF8().c_str() << endl;
         CCfVariant vtValue;
         pShcActionElement->GetValue(&vtValue);
         cout << "value = " << vtValue.uint32 << endl;
      }
}
```

ISHCActionTemplate

Description

The C++ interface "ISHCActionTemplate" specifies the methods of an action template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetName" method

Supplies the name of the "ISHCActionTemplate" instance.

```
CFRESULT GetName(CFSTR * pvarRet)
```

- `pvarRet`
  [out]: The name of the action template

"SetName" method

Sets the name of the "ISHCActionTemplate" instance

```
CFRESULT SetName(CFSTR  value)
```

- `value`
  [in]: The name

"GetDisplayNames" method

Supplies a map with the display names of the "ISHCActionTemplate" instance and their language code IDs.

```
CFRESULT GetDisplayNames)(ICfMapIDToVariant** ppDisplayNames)
```

- `ppDisplayNames`
  [out]: The map with String/String pairs (language code ID for display name).

"SetDisplayNames" method

Adds a display name and its language code ID to the map with the display name of the "ISHCActionTemplate" instance.

```
CFRESULT SetDisplayNames)(CFLCID languageId, CFSTR pDisplayName)
```

- `languageID`
  [in]: The language code ID of the display name
- `pDisplayName`
  [in]: The display name

"GetDescriptions" method

Supplies a map with the descriptions of the "ISHCActionTemplate" instance and their language code IDs.

```
CFRESULT GetDescriptions(ICfMapIDToVariant** value)
```

- `value`
  [out]: The map with String/String pairs (language code ID for description).

"SetDescriptions" method

Adds a description and its language code ID to the map with the descriptions of the "ISHCActionTemplate" instance.

```
CFRESULT SetDescription(CFLCID languageId, CFSTR pDescriptions)
```

- `languageID`
  [in]: The language code ID of the description
- `pDescriptions`
  [in]: The description text

"GetIsDeleted" method

Supplies the information on whether the "ISHCActionTemplate" instance was deleted by users.

```
CFRESULT GetIsDeleted(CFBOOL* pIsDeleted)
```

- `pIsDeleted`
  [out]:
  - 0: Was not deleted
  - 1: Was deleted

"GetElements" method

Supplies an "ISHCActionTemplateElementEnumerator" instance. The enumerator provides you with access to the "ISHCActionTemplateElement" instances of the "ISHCActionTemplate" instance.

```
CRFESULT GetElements(ISHCActionTemplateElementEnumerator**
ppISHCActionTemplateElementEnumerator)
```

- `ppISHCActionTemplateElementEnumerator`
  [out]: The enumerator

"CreateElement" method

Adds an "ISHCActionTemplateElement" instance to the "ISHCActionTemplate" instance.

```
CRFESULT CreateElement(ISHCActionTemplateElement*
pSHCActionTemplateElement)
```

- `pSHCActionTemplateElement`
  [in]: Reference to the new action element

"DeleteElement" method

Deletes an action element of the "ISHCActionTemplate" instance.

```
CRFESULT DeleteElement(ISHCActionTemplateElement*
pSHCActionTemplateElement)
```

- `pSHCActionTemplateElement`
  [in]: Reference to the action element to be deleted

See also

Locale IDs of the supported languages (Page 7829)

ISHCActionTemplateEnumerator

Description

The C++ interface "ISHCActionTemplateEnumerator" specifies methods for handling the enumeration of the action templates of an "ISHCCalendar" instance.

The enumeration is returned by the "Read" method of an "ISHCActionTemplatesProvider" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionTemplate** ppItem)
```

- `ppItem`
  [out]: The current action template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of action templates

Example

Copy code

```
void PrintActionTemplate(const ISHCActionTemplateEnumeratorPtr& p_pShcActionTemplateEnum)
{
   uint32_t nCount = 0;
   p_pShcActionTemplateEnum->Count(&nCount);
   for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
   {
      cout << endl;
      p_pShcActionTemplateEnum->MoveNext();
      ISHCActionTemplatePtr pShcActionTemplate;
      p_pShcActionTemplateEnum->Current(&pShcActionTemplate);
      CCfString strActionTemplateName;
      pShcActionTemplate->GetName(&strActionTemplateName);
      cout << "ActionTemplateName=" << strActionTemplateName.ToUTF8().c_str() << endl;

      CFBOOL bIsDeleted;
      pShcActionTemplate->GetIsDeleted(&bIsDeleted);
      cout << "IsDeleted=" << (uint32_t)bIsDeleted << endl;
      ICfMapIDToVariantPtr pDescriptions;
      pShcActionTemplate->GetDescriptions(&pDescriptions);
      std::cout << "Descriptions=" << std::endl << std::endl;
      uint32_t nCount1 = 0;
      pDescriptions->Count(&nCount1);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
      {
         int32_t nLanguageID;
         pDescriptions->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strDescription;
         pDescriptions->ValueAt(nLanguageID, &strDescription);
         std::cout << "LangauageID =" << nLanguageID << " Description=" <<
CCfSmartString(strDescription).ToUTF8().c_str() << std::endl;
      }

      ICfMapIDToVariantPtr pDisplayNames;
      pShcActionTemplate->GetDisplayNames(&pDisplayNames);
      std::cout << "DisplayNames::" << std::endl << std::endl;
      uint32_t nCount1 = 0;
      pDisplayNames->Count(&nCount1);
      for (uint32_t nIndex1 = 0; nIndex1 < nCount1; nIndex1++)
      {
         int32_t nLanguageID;
         pDisplayNames->KeyAt(nIndex1, &nLanguageID);
         CCfVariant strDIsplayname;
         pDisplayNames->ValueAt(nLanguageID, &strDIsplayname);
         std::cout << "LangauageID =" << nLanguageID << " DisplayName=" <<
CCfSmartString(strDIsplayname).ToUTF8().c_str() << std::endl;
      }

      ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
      pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);

      uint32_t nCount2 = 0;
      pShcActionTemplateElementEnum   ->Count(&nCount2);
      for (uint32_t nIndex = 0; nIndex < nCount2; nIndex++)
      {
```

Copy code

```
          pShcActionTemplateElementEnum   ->MoveNext();
          ISHCActionTemplateElementPtr pShcActionTemplateElement;
          pShcActionTemplateElementEnum   ->Current(&pShcActionTemplateElement);
          CCfString strElementName;
          pShcActionTemplateElement->GetElementName(&strElementName);
          cout << "TemplateElementName= " << strElementName.ToUTF8().c_str() << endl;
          CFENUM type;
          pShcActionTemplateElement->GetElementType(&Type);
          cout << "ElementType=" << Type << endl;
          CCfTimeSpan64 tsOffset;
         cout << "Element Offset=" << tsOffset.GetTimeSpanString().ToUTF8().c_str() << endl;
          CCfVariant vtValue;
          pShcActionTemplateElement->GetValue(&vtValue);
          cout << "Value=" << vtValue.uint32 << endl;
       }
    }
}
```

ISHCActionTemplatesProvider

Description

The C++ interface "ISHCActionTemplatesProvider" provides you with access to an "ISHCActionTemplateEnumerator" instance which contains an enumeration with the action templates of an "ISHCCalendar" instance. With the methods of the provider, you can create, read, update and delete action templates. The provider is returned by the "GetActionTemplatesProvider" method of an "ISHCCalendar" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"Browse" method

Supplies an "ISHCActionTemplateEnumerator" instance which has access to an enumeration with the "ISHCActionTemplate" instances of the calendar.

```
CRFESULT Browse(CFBOOL includeDeleted,OUT
ISHCActionTemplateEnumerator** ppISHCActionTemplateEnumerator)
```

- `includeDeleted`
  Saves whether the enumerator also has access to the deleted action templates.
- `ppISHCActionTemplateEnumerator`
  [out]: The enumerator

Example:

```
ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
pShcActionTemplateProvider->Browse(CF_FALSE,
&pShcActionTemplateEnum);
```

"Create" method

Adds new action templates to the enumeration with the "ISHCActionTemplate" instances of the calendar.

```
CFRESULT Create(ICfArrayIUnknown* pActionTemplates)
```

- `pActionTemplates`
  [in]: An array with the action templates to be added.

Example:

Copy code

```
void CreateActionTemplateWithActionTemplateElement()
{
   ICfUnknownPtr pUnk;
   m_pCalendar->GetObject(ODK_SHC_ACTION_TEMPLATE, &pUnk);
   CCfArrayIUnknown ArrayTemplate;
   ICfArrayIUnknownPtr pArrayTemplate;
   ISHCActionTemplatePtr pShcActionTemplate = (ISHCActionTemplatePtr)pUnk;
   pShcActionTemplate->SetName(CCfString(L"ActionTemplateName"));
   pShcActionTemplate->SetDisplayName(1033, CCfString(L"ActionTemplateDisplayName"));
   pShcActionTemplate->SetDescription(1033, CCfString(L"ActionTemplateDescription"));
   ArrayTemplate.Append(pShcActionTemplate);
   ArrayTemplate.DetachEnumerator(&pArrayTemplate);
   m_pShcActionTemplateProvider->Create(pArrayTemplate);
   m_pCalendar->GetObject(ODK_SHC_ACTION_TEMPLATE_ELEMENT, &pUnk);
   ISHCActionTemplateElementPtr pShcActionTemplateElement =
(ISHCActionTemplateElementPtr)pUnk;
   pShcActionTemplateElement->SetElementName(CCfString(L"HMI_RT_1::Unit1.Member_1"));
   CFTIMESPAN64 Offset = Get1Hour();
   pShcActionTemplateElement->SetOffset(Offset);
   uint32_t value = 1;
   pShcActionTemplateElement->SetValue(CCfVariant(value));
   pShcActionTemplate->CreateElement(pShcActionTemplateElement);
}
```

"Update" method

Updates "ISHCActionTemplate" instances of the enumeration.

```
CRFESULTUpdate(ICfArrayIUnknown* pActionTemplates)
```

- `pActionTemplates`
  [in]: An array with the action templates to be updated.

Example:

Copy code

```
void UpdateActionTemplateWithActionTemplateElement()
{
   ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
   m_pShcActionTemplateProvider->Browse(CF_FALSE, &pShcActionTemplateEnum);
   pShcActionTemplateEnum->MoveNext();
   CCfArrayIUnknown ArrayActionTemplate;
   ICfArrayIUnknownPtr pArrayActionTemplate;
   ISHCActionTemplatePtr pShcActionTemplate;
   pShcActionTemplateEnum->Current(&pShcActionTemplate);
   pShcActionTemplate->SetName(CCfString(L"UpdatedActionTemplate"));
   pShcActionTemplate->SetDisplayName(1033,
CCfString(L"UpdatedActionTemplateDisplayName"));
   ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
   pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
   pShcActionTemplateElementEnum->MoveNext();
   ISHCActionTemplateElementPtr pShcActionTemplateElement;
   pShcActionTemplateElementEnum->Current(&pShcActionTemplateElement);
   CCfTimeSpan64 offset = Get1Hour() * 2;
   pShcActionTemplateElement->SetOffset(offset);
   uint32_t nValue = 2;
   pShcActionTemplateElement->SetValue(CCfVariant(nValue));

   ArrayActionTemplate.Append(pShcActionTemplate);
   ArrayActionTemplate.DetachEnumerator(&pArrayActionTemplate);
   m_pShcActionTemplateProvider->Update(pArrayActionTemplate);
}
```

"Delete" method

Deletes "ISHCActionTemplate" instances of the calendar from the enumeration.

```
CRFESULT Delete(ICfArrayIUnknown* pActionTemplates)
```

- `pActionTemplates`
  [in]: An array with the action templates to be deleted.

Example:

Copy code

```
void DeleteActionTemplateWithActionTemplateElement()
{
   CFRESULT hr = CF_ERROR;
   ISHCActionTemplateEnumeratorPtr pShcActionTemplateEnum;
   m_pShcActionTemplateProvider->Browse(CF_FALSE, &pShcActionTemplateEnum);
   pShcActionTemplateEnum->MoveNext();
   CCfArrayIUnknown ArrayActionTemplate;
   ICfArrayIUnknownPtr pArrayActionTemplate;
   ISHCActionTemplatePtr pShcActionTemplate;
   pShcActionTemplateEnum->Current(&pShcActionTemplate);
   ISHCActionTemplateElementEnumeratorPtr pShcActionTemplateElementEnum;
   pShcActionTemplate->GetElements(&pShcActionTemplateElementEnum);
   pShcActionTemplateElementEnum->MoveNext();
   ISHCActionTemplateElementPtr pShcActionTemplateElement;
   pShcActionTemplateElementEnum->Current(&pShcActionTemplateElement);
   pShcActionTemplate->DeleteElement(pShcActionTemplateElement);

   ArrayActionTemplate.Append(pShcActionTemplate);
   ArrayActionTemplate.DetachEnumerator(&pArrayActionTemplate);
   m_pShcActionTemplateProvider->Delete(pArrayActionTemplate);
}
```

ISHCActionTemplateElement

Description

The C++ interface "ISHCActionTemplateElement" specifies the methods of an action template.

The interface inherits from the "ICfUnknown" interface.

Members

"GetElementType" method

Supplies the type of the "ISHCActionTemplateElement" instance.

`CRFESULT GetElementType(CFENUM* pElementType)`

- `pElementType`
  [out]: Points to the enumeration "ShcActionElementType", which can contain the following values:
  - `Tag` (0)
    The action element controls a tag.

"GetOffset" method

Returns the offset of the "ISHCActionTemplateElement" instance in relation to the anchor point of its "ISHCActionTemplate" instance.

```
CFRESULT GetOffset(CFTIMESPAN64* pOffset)
```

- `pOffset`
  [out]: The offset in 100 nanoseconds. Positive and negative value allowed.

"SetOffset" method

Sets the offset of the "ISHCActionTemplateElement" instance in relation to the anchor point of its "ISHCActionTemplate" instance.

```
CFRESULT SetOffset(CFTIMESPAN64 offset)
```

- `offset`
  [in]: The offset in 100 nanoseconds. Positive and negative value allowed.

"GetElementName" method

Supplies the name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT GetElementName(CFSTR* pElementName)
```

- `pElementName`
  [out]: The tag name

"SetElementName" method

Sets the name of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT SetElementName(CFSTR pElementName)
```

- `pElementName`
  [in]: The tag name

"GetValue" method

Supplies the value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CFRESULT GetValue(CFVARIANT* pValue)
```

- `pValue`
  [out]: The tag value

"SetValue" method

Sets the value of the tag controlled by the "ISHCActionTemplateElement" instance.

```
CRFESULT SetValue(CFVARIANT pValue)
```

- `pValue`
  [in]: The new tag value

ISHCActionTemplateElementEnumerator

Description

The C++ interface "ISHCActionTemplateElementEnumerator" specifies methods for handling the enumeration of "ISHCActionTemplateElement" instances of an "ISHCActionTemplate" instance.

The enumeration is returned by the "GetElements" method of an "ISHCActionTemplate" instance.

The interface inherits from the "ICfUnknown" interface.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ISHCActionTemplateElement** ppItem)
```

- `ppItem`
  [out]: The current action element of the action template

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of "ISHCActionTemplateElement" instances

Example

Copy code

```
void PrintActionTemplateElement(const ISHCActionTemplateElementEnumeratorPtr&
pShcActionTemplateElementEnum   )
{
   uint32_t nCount = 0;
   pShcActionTemplateElementEnum   ->Count(&nCount);
   for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
   {
      pShcActionTemplateElementEnum   ->MoveNext();
      ISHCActionTemplateElementPtr pShcActionTemplateElement;
      pShcActionTemplateElementEnum   ->Current(&pShcActionTemplateElement);
      CCfString strElementName;
      pShcActionTemplateElement->GetElementName(&strElementName);
      cout << "TemplateElementName= " << strElementName.ToUTF8().c_str() << endl;
      CFENUM type;
      pShcActionTemplateElement->GetElementType(&Type);
      cout << "ElementType=" << Type << endl;
      CCfTimeSpan64 tsOffset;
      cout << "Element Offset=" << tsOffset.GetTimeSpanString().ToUTF8().c_str() << endl;
      CCfVariant vtValue;
      pShcActionTemplateElement->GetValue(&vtValue);
      cout << "Value=" << vtValue.uint32 << endl;
   }
}
```

20.3.9.8 Interfaces of the contexts

IContextLogging

Description

The C++ interface "IContextLogging" defines events and methods for creating and reading "IContextDefinition" instances as well as for starting, stopping, monitoring and reading their "ILoggedContext" instances. You can use "ILoggedContext" instances to filter runtime data, for example, for alarms that fall within the time period of a particular "ILoggedContext" instance.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"CreateContextDefinitions" method

Creates ContextDefinitions in the database.

```
CRFESULT CreateContextDefintions(ICfArrayIUnknown*
pContextDefinitions, IContextLoggingCallBack *pContextCallBack)
```

- `pContextDefinitions`:
  [in]: Collection with "IContextDefinition" instances
- `*pContextCallBack`:
  [in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.

"ReadContextDefinitions" method

Reads ContextDefinitions from the database. The instances can be filtered by plant object and HMIContextProviderType .

```
CRFESULT ReadContextDefinitions(IContextLoggingCallBack*
pContextCallBack, ICfArrayString* plantViewPaths, ICfArrayVariant*
pProviderTypes, CFENUM sortingMode)
```

- `pContextCallBack`:
  [in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.
- `plantViewPaths`:
  [in]: Limits the read operation to "IContextDefinition" instances from this collection of plant objects.
- `pProviderTypes`:
  [in/optional]: Limits the read operation to "IContextDefinition" instances with HmiContextProvider types from this collection.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `sortingMode`:
  [in]: Points to the enumeration HmiContextLoggingSortingMode, which can contain the following values:
  - Ascending = 1
    Default setting
  - Descending = 2

"ReadContexts" method

Reads "ILoggedContext" instances of a specific time period. The instances can be filtered using a "IContextFilter" instance.

```
CRFESULT ReadContexts(CFDATETIME64 start, CFDATETIME64
end, IContextLoggingCallBack* ContextCallBack, IContextFilter*
FilterObject, CFENUM type)
```

- `start`:
  [in]: The start time of the period within which the "ILoggedContext" instances must lie.
- `end`:
  [in]: The end time of the period within which the "ILoggedContext" instances must lie.
- `ContextCallBack`:
  [in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.
- `FilterObject`:
  [in/optional]: The "IContextFilter" instance whose filter settings are used.
- `type`:
  [in]: Points to the enumeration HmiContextLoggingSortingMode, which can contain the following values:
  - Ascending = 1
    Default setting
  - Descending = 2

"StartContext" method

Creates a new "ILoggedContext" instance for a "IContextDefinition" instance.

```
CRFESULT StartContext(CFSTR p_strContextName, CFENUM providerType,
CFSTR plantViewPath, CFVARIANT contextValue, CFDATETIME64
startTime, uint32_t qualityCode)
```

- `p_strContextName`:
  [in]: The name of the "IContextDefinition" instance for which the context log entry is created.
- `providerType`:
  [in]: The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `plantViewPath`:
  [in]: Path to an "IPlantObject" instance
  E.g.: ".hierarchy::PlantView/Ln_1/M_1"
  Is used in combination with `p_strContextName` for unique identification of the "IContextDefinition" instance.

- `contextValue`:
  [in]: The context value of the context log entry
- `startTime`:
  [in]: Start time of the new context log entry
- `qualityCode`:
  [in]: The QualityCode of the context value of the context log entry

"StopContext" method

Stops the currently running "ILoggedContext" instance of an "IContextDefinition" instance.

```
CRFESULT StopContext(CFSTR p_strContextName, CFENUM providerType,
CFSTR plantViewPath, CFDATETIME64 endtime)
```

- `p_strContextName`:
  [in]: The name of the "IContextDefinition" instance whose context log entry is stopped.
- `providerType`:
  [in]: The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `plantViewPath`:
  [in]: Path to an "IPlantObject" instance
  E.g.: ".hierarchy::PlantView/Ln_1/M_1"
  Is used in combination with `p_strContextName` for unique identification of the "IContextDefinition" instance.
- `endtime`:
  [in]: End time of the new context log entry

"Add" method

Adds a "IContextDefintion" instance to a vector. The methods Clear(), Subscribe() and CancelSubscription() can be called for the instances of the vector.

```
CRFESULT Add(CFSTR contextName, CFENUM type, CFSTR plantViewPath)
```

- `contextName`:
  [in]: The name of the "IContextDefintion" instance
- `type`:
  [in]: The HmiContextProviderType of the "IContextDefintion" instance
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.
- `plantViewPath`:
  [in]: Path to an "IPlantObject" instance
  E.g.: ".hierarchy::PlantView/Ln_1/M_1"
  Is used in combination with `contextName` for unique identification of the "IContextDefinition" instance.

"Clear" method

Deletes the "IContextDefintion" instances added via `Add()` from the vector.

```
CRFESULT Clear()
```

"Subscribe" method

Subscribes the "IContextDefinition" instances added to the vector via `Add()` for monitoring.

```
CRFESULT Subscribe(IContextLoggingCallBack* pContextLoggingCallback)
```

- `pContextLoggingCallback`:
  [in]: Points to the "IContextLoggingCallBack" object that implements the callback interface.

"CancelSubscribe" method

Unsubscribes the "IContextDefinition" instances added to the vector with `Add()` from monitoring.

```
CRFESULT CancelSubscribe()
```

Examples

Copying code

```
void CreateContextDefinitions(IRuntimePtr pRuntime)
{
   std::cout << std::endl << __FUNCTION__ << std::endl << std::endl;
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
   IContextLoggingPtr pContextLoggingPtr(pUnk);

   std::vector<IContextDefinitionPtr> vecContextDefinitions;
   for (size_t index = 0; index < 10; index++)
   {
      ICfUnknownPtr pUnkCd;
      pRuntime->GetObject(CCfString(L"ContextDefinition"), &pUnkCd);
      IContextDefinitionPtr pContextDefinition(pUnkCd);
      CCfSmartString strID = std::to_string(std::rand());
      CCfSmartString strName(L"CD-");
      strName.Append(strID);
      ICfMapIDToVariantPtr pDisplayNames;
      std::map<int32_t, CCfVariant> DisplayNames;
      CCfSmartString strDeuName(L"Deutsch-");
      strDeuName.Append(strName);
      DisplayNames[1031] = strDeuName;
      CCfSmartString strEngName(L"English-");
      strEngName.Append(strName);
      DisplayNames[1033] = strEngName;
      CCfMapIDToVariant::CreateEnumerator(DisplayNames, &pDisplayNames);
      pContextDefinition->SetDisplayNames(pDisplayNames);

      CCfString str_Name = L".hierarchy::Plant/Node1_1";
      pContextDefinition->SetPlantViewPath(str_Name);

      CCfString m_strName = L"Contetx_";
      m_strName.Append(strName.AllocCFSTR());
      pContextDefinition->SetName(m_strName);

      pContextDefinition->SetDataType(HmiContextDataType::String);
      vecContextDefinitions.push_back(pContextDefinition);

   }
   ICfArrayIUnknownPtr pContextDefnitionsArray;
   ::CfCreateEnumerator(vecContextDefinitions, &pContextDefnitionsArray);
   CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
   pContextLoggingCB->AddRef();
  pContextLoggingPtr->CreateContextDefintions(pContextDefnitionsArray, pContextLoggingCB);
}

void DisplayContextError(const std::vector<IContextErrorPtr>& pVecContext)
{
   for (const auto& pContext : pVecContext)
   {
      CCfString strContextName;
      pContext->GetContextName(&strContextName);
      uint32_t value;
      pContext->GetErrorCode(&value);
   }
```

Copying code

```
}
```

Copying code

```
void ReadContextDefinitionsWithFilter(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
   IContextLoggingPtr pContextLoggingPtr(pUnk);
   CCfString str_Name = L".hierarchy::Plant/Node1_1";
   ICfArrayStringPtr pArrayPlantObjects;
   CCfArrayString ArrayPlantobjects;
   ArrayPlantobjects.Append(CCfVariant(str_Name));
   ArrayPlantobjects.DetachEnumerator(&pArrayPlantObjects);
   CCfVariant vtProviderTypes(static_cast<CFENUM>(HmiContextProviderType::UserDefined));
   CCfArrayVariant arryProvidertypes;
   arryProvidertypes.Append(vtProviderTypes);
   ICfArrayVariantPtr pArrayProvidertypes;
   arryProvidertypes.DetachEnumerator(&pArrayProvidertypes);
   CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
   pContextLoggingCB->AddRef();
   pContextLoggingPtr->ReadContextDefinitions(pContextLoggingCB, pArrayPlantObjects,
pArrayProvidertypes);

   pContextLoggingCB->Release();
}


void DisplayContextDefinition(const std::vector<IContextDefinitionPtr>& ContextDef)
{
   for (auto& pValues : ContextDef)
   {
      uint32_t ErrorCode;
      pValues->GetErrorCode(&ErrorCode);
      CCfString strName;
      pValues->GetName(&strName);
      CFENUM pnum;
      pValues->GetProviderType(&pnum);
      HmiContextDataType dataType;
      pValues->GetDataType(&dataType);
      ICfMapIDToVariantPtr displayName;
      pValues->GetDisplayNames(&displayName);
      uint32_t nCount;
      displayName->Count(&nCount);
      for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
      {
         int32_t langID;
         displayName->KeyAt(index, &langID);
         CCfVariant vtName;
         displayName->ValueAt(langID, &vtName);
      }
      CCfString strPlantViewPath;
      pValues->GetPlantViewPath(&strPlantViewPath);
   }
}
```

Copying code

```
void StartContext(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
   IContextLoggingPtr pContextLoggingPtr(pUnk);
   CCfDateTime64 dtStartTime = CCfDateTime64::Now();
   CCfVariant vtValue = L"Orange Juice";
   uint32_t nQuality = 192;
   CCfString strNameContextName = L"ContetxName_";
   strNameContextName.Append(strContext_Unique_Num);
   pContextLoggingPtr->StartContext(strNameContextName,
static_cast<CFENUM>(HmiContextProviderType::UserDefined), CCfString(L".hierarchy::Plant/
Node1_1"),vtValue, dtStartTime, nQuality);
}
```

Copying code

```
void StopContext(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
   IContextLoggingPtr pContextLoggingPtr(pUnk);
   CCfDateTime64 dtEndTime = CCfDateTime64::Now();
   CCfString strNameContextName = L"ContetxName_";
   strNameContextName.Append(strContext_Unique_Num);
   pContextLoggingPtr->StopContext(strNameContextName,
static_cast<CFENUM>(HmiContextProviderType::UserDefined), CCfString(L".hierarchy::Plant/
Node1_1"), dtEndTime);
}
```

Copying code

```
void ReadContextWithFilter(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   IContextLoggingPtr pContextLoggingPtr(pUnk);
   CCfDateTime64 dtEndTime = CCfDateTime64::Now();
   CCfDateTime64 dtStartTime;
   IContextFilterPtr pFilter;
   ICfUnknownPtr pUnkFilter;
   pRuntime->GetObject(CCfString(L"ContextFilter"), &pUnkFilter);
   pFilter = pUnkFilter;
   CCfString strNameContextName = L"ContetxName_";
   strNameContextName.Append(strContext_Unique_Num);
   pFilter->SetContextName(strNameContextName);
   pFilter->SetProviderType(static_cast<CFENUM>(HmiContextProviderType::UserDefined));
   pFilter->SetOperator(CCfString(L"="));
   pFilter->SetValue(CCfVariant(L"Orange Juice"));
   pFilter->SetPlantViewPath(CCfString(L".hierarchy::Plant/Node1_1"));
   CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
   pContextLoggingCB->AddRef();
   pContextLoggingPtr->ReadContexts(dtStartTime, dtEndTime, pContextLoggingCB, pFilter,
HmiContextLoggingSortingMode::Ascending);
   pContextLoggingCB->WaitForcompletion(MaxContextWaitTime);
   std::vector<ILoggedContextPtr> Context = pContextLoggingCB->GetContexts();
                  DisplayContext(Context);
}

void DisplayContext(const std::vector<ILoggedContextPtr>& Context)
{
   for (const auto& item : Context)
   {
      uint32_t pError;
      item->GetErrorCode(&pError);
      CCfDateTime64 dtStartTime;
      item->GetStartTime(&dtStartTime);
      CCfString strStartTime = dtStartTime.GetDateTimeString();
      CCfDateTime64 dtEnd;
      item->GetEndTime(&dtEnd);
      CCfString strEndTime = dtEnd.GetDateTimeString();
      uint32_t pQuality;
      item->GetQuality(&pQuality);
      CCfString name;
      item->GetName(&name);
      CCfString viewPath;
      item->GetPlantViewPath(&viewPath);
      CFENUM providerType;
      item->GetProviderType(&providerType);
      CCfVariant vtValue;
      item->GetValue(&vtValue);
      PrintVariantType(vtValue);
   }
}
```

Copying code

```
void SubscribeContextLogging(IRuntimePtr pRuntime)
{
   ICfUnknownPtr pUnk;
   pRuntime->GetObject(CCfString(L"ContextLogging"), &pUnk);
   IContextLoggingPtr pContextLoggingPtr(pUnk);
   CContextLoggingCB * pContextLoggingCB = new CContextLoggingCB();
   pContextLoggingCB->AddRef();
   CreateContextDefinition(pRuntime);
   CCfString strNameContextName = L"ContetxName_";
   strNameContextName.Append(strContext_Unique_Num);
   CFENUM providerType = static_cast<CFENUM>(HmiContextProviderType::UserDefined);
   pContextLoggingPtr->Add(strNameContextName, providerType, CCfString(L".hierarchy::Plant/
Node1_1"));
   pContextLoggingPtr->Subscribe(pContextLoggingCB);
   StartContext(pRuntime);
   if (pContextLoggingCB->GetContexts().size()>0 || pContextLoggingCB-
>WaitForcompletion(MaxContextWaitTime) == CF_SUCCESS);
   {
      std::vector<ILoggedContextPtr> Context = pContextLoggingCB->GetContexts();
      DisplayContext(Context);
      pContextLoggingCB->clear();
   }
   StopContext(pRuntime);
   if (pContextLoggingCB->GetContexts().size() > 0 || pContextLoggingCB-
>WaitForcompletion(MaxContextWaitTime)==CF_SUCCESS);
   {
      std::vector<ILoggedContextPtr> Context = pContextLoggingCB->GetContexts();
      DisplayContext(Context);
   }
   pContextLoggingPtr->CancelSubscribe();
   pContextLoggingCB->Release();
}

void DisplayContext(const std::vector<ILoggedContextPtr>& Context)
{
   for (const auto& item : Context)
   {
      uint32_t pError;
      item->GetErrorCode(&pError);
      CCfDateTime64 dtStartTime;
      item->GetStartTime(&dtStartTime);
      CCfString strStartTime = dtStartTime.GetDateTimeString();
      CCfDateTime64 dtEnd;
      item->GetEndTime(&dtEnd);
      CCfString strEndTime = dtEnd.GetDateTimeString();
      uint32_t pQuality;
      item->GetQuality(&pQuality);
      CCfString name;
      item->GetName(&name);
      CCfString viewPath;
      item->GetPlantViewPath(&viewPath);
      CFENUM providerType;
      item->GetProviderType(&providerType);
      CCfVariant vtValue;
      item->GetValue(&vtValue);
```

Copying code

```
        PrintVariantType(vtValue);
    }
}
```

IContextLoggingCallBack

Description

The C++ interface "IContextLoggingCallBack" defines methods for implementing asynchronous operations for monitoring "IContextDefinition" instances. The methods are used by the "IContextLogging" interface.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"OnCreate" method

Callback method is called when a ContextDefintion is created in the database by calling the "IContextLogging.CreateContextDefinitions" method.

```
CFRESULT OnCreate(uint32_t globalError, CFSTR SystemName,
IContextErrorEnumerator* errors, CFBOOL Completed)
```

- `globalError`:
  [in]: Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
- `SystemName`:
  [in]: Name of the system on which the ContextDefinitions have been created.
- `errors`:
  [in]: Enumerator for the instance-specific errors that were created when the ContextDefinitions were created.
- `Completed`:
  [in]: Status of the asynchronous transfer:
  - True: All ContextDefinitions have been notified.
  - False: Additional notifications are expected.

Members "OnRead"

Callback method is called:

- When a ContextDefintion is read from the database by calling the "IContextLogging.ReadContextDefinitions" method.
  The method with the following signature is called:
  `CFRESULT OnRead(uint32_t globalError, CFSTR SystemName, IContextDefinitionEnumerator* contextLoggingResult, CFBOOL Completed)`
  - `globalError`:
    [in]: Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
  - `SystemName`:
    [in]: Name of the system on which the ContextDefinitions have been created.
  - `contextLoggingResult`:
    [in]: Enumerator for the read "IContextDefinition" instances
  - `Completed`:
    [in]: Status of the asynchronous transfer:
    True: All ContextDefinitions have been notified.
    False: Additional notifications are expected.
- When a LoggedContext is read by calling the "IContextLogging.ReadContexts" method.
  The method with the following signature is called:
  `CFRESULT OnRead(uint32_t globalError, CFSTR SystemName, ILoggedContextEnumerator* loggedContexts, CFBOOL Completed)`
  - `globalError`:
    [in]: Global ErrorCode when a global error occurs during the call. All other parameters are invalid in this case.
  - `SystemName`:
    [in]: Name of the system on which the "ILoggedContext" instances have been created.
  - `contextLoggingResult`:
    [in]: Enumerator for the read "ILoggedContext" instances that were started or stopped.
  - `Completed`:
    [in]: Status of the asynchronous transfer:
    True: All ContextDefinitions have been notified.
    False: Additional notifications are expected.
  - `Completed`:
    [in]: Status of the asynchronous transfer:
    True: All ContextDefinitions have been notified.
    False: Additional notifications are expected.

"OnDataChanged" method

Callback method is called when a monitored "ILoggedContext" instance is started or stopped by calling "IContextLogging.StartContext" or "IContextLogging.StopContext".

```
CFRESULT OnDataChanged(ILoggedContextEnumerator* pEnumerator)
```

- `pEnumerator`:
  [out]: Points to an "ILoggedContextEnumerator" object which contains an enumeration with "ILoggedContext" instances.

IContextDefinitionEnumerator

Description

The C++ interface "IContextDefinitionEnumerator" specifies methods for handling the enumeration of "IContextDefinition" instances. The enumeration is returned by the "OnRead" method of an "IContextLoggingCallBack" instance.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(IContextDefinition **pItem)
```

- `ppItem`
  [out]: The current "IContextDefinition" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of "IContextDefinition" instances

IContextErrorEnumerator

Description

The C++ interface "IContextErrorEnumerator" specifies methods for handling the enumeration of "IContextError" instances. The enumeration is returned by the "OnCreate" method of an "IContextLoggingCallBack" instance.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(IContextError **pItem)
```

- `ppItem`
  [out]: The current "IContextError" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of "IContextError" instances

ILoggedContextEnumerator

Description

The C++ interface "ILoggedContextEnumerator" specifies methods for handling the enumeration of "ILoggedContext" instances. The enumeration is returned by the "OnRead" method of an "IContextLoggingCallBack" instance.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"MoveNext" method

Go to the next element of the enumeration.

```
CFRESULT MoveNext()
```

"Current" method

Output the current element of the enumeration.

```
CFRESULT Current(ILoggedContext **pItem)
```

- `ppItem`
  [out]: The current "ILoggedContext" instance

"Reset" method

Reset the current position in the enumeration. The "MoveNext" method moves afterwards to the first element.

```
CFRESULT Reset()
```

"Count" method

Output the size of the enumeration or the number of its elements.

```
CRFESULT Count(uint32_t* pCount)
```

- `pCount`
  [out]: Number of "ILoggedContext" instances

IContextDefinition

Description

The C++ interface "IContextDefinition" specifies properties for defining "IContextDefinition" instances. "ILoggingContext" instances can be created using the StartContext() and StopContext() methods based on an "IContextDefiniton" instance.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetPlantViewPath" method

Sets the path to the plant object of the "IContextDefinition" instance.

```
CRFESULT GetPlantViewPath(CFSTR* p_strPlantViewPath)
```

- p_strPlantViewPath:
  [out]: The path to the plant object
  Example: ".hierarchy::Plant/Station"

"SetPlantViewPath" method

Sets the path to the plant object of the "IContextDefinition" instance.

```
CRFESULT SetPlantViewPath(CFSTR p_strPlantViewPath)
```

- p_strPlantViewPath:
  [in]: The path to the plant object
  Example: ".hierarchy::Plant/Station"

"GetProviderType" method

The source that creates the instance.

Is used together with "Name" to uniquely identify a "IContextDefinition" instance.

```
CRFESULT GetProviderType(CFENUM* pContextType)
```

- pContextType:
  [out]: The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"GetName" method

Supplies the name of the "IContextDefinition" instance.

```
CRFESULT GetName(CFSTR* pName)
```

- pName:
  [out]: The name

"SetName" method

Sets the name of the "IContextDefinition" instance.

Sets the name of the "IContextDefinition" instance.

```
CRFESULT SetName(IN CFSTR name)
```

- name:
  [in]: The name

"GetDisplayNames" method

Supplies the display name of the "IContextDefinition" instance

```
CRFESULT GetDisplayNames(ICfMapIDToVariant** ppDisplayName)
```

- `ppDisplayName`:
  [out]: The display name

"SetDisplayNames" method

Sets the display name of the "IContextDefinition" instance.

```
CRFESULT SetDisplayNames(ICfMapIDToVariant* pDisplayName)
```

- `pDisplayName`:
  [in]: The display name

"GetDataType" method

Supplies the data type of the "IContextDefinition" instance.

```
CRFESULT GetDataType(HmiContextDataType* pdatatype)
```

- `pdatatype`:
  [out]: The data type
  The enumeration "HmiContextDataType" can contain the following values:
  - Bool = 0x01
  - SInt = 0x02
  - Int = 0x03
  - DInt = 0x04
  - LInt = 0x05
  - USInt = 0x06
  - UInt = 0x07
  - UDInt = 0x08
  - ULInt = 0x09
  - Real = 0x0A
  - LReal = 0x0B
  - LTime = 0x0C
  - DateTime = 0x0D
  - Byte = 0x11
  - Word = 0x12
  - DWord = 0x13
  - LWord = 0x14
  - String = 0x32

"SetDataType" method

Sets the data type of the "IContextDefinition" instance.

```
CRFESULT SetDataType(HmiContextDataType datatype)
```

- `datatype`:
  [in]: The data type
  The enumeration "HmiContextDataType" can contain the following values:
  - Bool = 0x01
  - SInt = 0x02
  - Int = 0x03
  - DInt = 0x04
  - LInt = 0x05
  - USInt = 0x06
  - UInt = 0x07
  - UDInt = 0x08
  - ULInt = 0x09
  - Real = 0x0A
  - LReal = 0x0B
  - LTime = 0x0C
  - DateTime = 0x0D
  - Byte = 0x11
  - Word = 0x12
  - DWord = 0x13
  - LWord = 0x14
  - String = 0x32

"GetErrorCode" method

Supplies the error code of the "IContextDefinition" instance

```
CRFESULT GetErrorCode(uint32_t* error)
```

- `error`:
  [out]:

Examples

Copy code

```
void DisplayContextDefinition(const std::vector<IContextDefinitionPtr>& ContextDef)
{
   for (auto& pValues : ContextDef)
   {
      uint32_t ErrorCode;
      pValues->GetErrorCode(&ErrorCode);
      CCfString strName;
      pValues->GetName(&strName);
      CFENUM pnum;
      pValues->GetProviderType(&pnum);
      HmiContextDataType dataType;
      pValues->GetDataType(&dataType);
      ICfMapIDToVariantPtr displayName;
      pValues->GetDisplayNames(&displayName);
      uint32_t nCount;
      displayName->Count(&nCount);
      for (uint32_t nIndex = 0; nIndex < nCount; nIndex++)
      {
         int32_t langID;
         displayName->KeyAt(index, &langID);
         CCfVariant vtName;
         displayName->ValueAt(langID, &vtName);
      }
      CCfString strPlantViewPath;
      pValues->GetPlantViewPath(&strPlantViewPath);
   }
}
```

ILoggedContext

Description

The C++ interface "ILoggedContext" defines properties of context log entries of an "IContextDefinition" instance.

The context log entries are started and stopped using methods of the "IContextLogging" interface.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetStartTime" method

Supplies the start time of the "IContextLogging" instance.

```
CRFESULT GetStartTime(CFDATETIME64* value)
```

- `value`:
  [out]: The start time

"GetEndTime" method

Supplies the end time of the "IContextLogging" instance.

```
CRFESULT GetEndTime(CFDATETIME64* value)
```

- `value`:
  [out]: The end time

"GetErrorCode" method

Supplies the error code of the "IContextLogging" instance.

```
CRFESULT GetErrorCode(uint32_t* error)
```

- `error`:
  [out]: The error code

"GetValue" method

Supplies the value of the "IContextLogging" instance. The value has the same data type as is specified by the "DataType" property of the "IContextDefinition" instance.

Example: An "IContextDefinition" instance has the name "Product" and the data type String. Its "IContextLogging" instance has the value "Limo".

```
CRFESULT GetValue(CFVARIANT * value)
```

- `value`:
  [out]: The name of the "IContextDefinition" instance in the user interface

"GetQuality" method

Supplies the QualityCode of the context value.

```
CRFESULT GetQuality(uint32_t* quality)
```

- `quality`:
  [out]: The QualityCode

Examples

Copy code

```
void DisplayContext(const std::vector<ILoggedContextPtr>& Context)
{
   for (const auto& item : Context)
   {
      uint32_t pError;
      item->GetErrorCode(&pError);
      CCfDateTime64 dtStartTime;
      item->GetStartTime(&dtStartTime);
      CCfString strStartTime = dtStartTime.GetDateTimeString();
      CCfDateTime64 dtEnd;
      item->GetEndTime(&dtEnd);
      CCfString strEndTime = dtEnd.GetDateTimeString();
      uint32_t pQuality;
      item->GetQuality(&pQuality);
      CCfString name;
      item->GetName(&name);
      CCfString viewPath;
      item->GetPlantViewPath(&viewPath);
      CFENUM providerType;
      item->GetProviderType(&providerType);
      CCfVariant vtValue;
      item->GetValue(&vtValue);
      PrintVariantType(vtValue);
   }
}
```

IContextError

Description

The C++ interface "IContextError" specifies methods for accessing error results that occur when creating ContextDefinitions in the database.

The interface implements the methods of the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetContextName" method

Supplies the name of the "IContextDefintion" instance.

```
CRFESULT GetContextName(CFSTR* p_strName)
```

- `p_strName`:
  [out]: The name

"GetContextErrorCode" method

Supplies the error code.

```
CRFRESULT GetErrorCode)(OUT uint32_t* p_ErrorCode)
```

- `p_ErrorCode`:
  [out]: The error code

Examples

Copy code

```
void DisplayContextError(const std::vector<IContextErrorPtr>& pVecContext)
{
   for (const auto& pContext : pVecContext)
   {
      CCfString strContextName;
      pContext->GetContextName(&strContextName);
      uint32_t value;
      pContext->GetErrorCode(&value);
}
```

IContextFilter

Description

The C++ interface "IContextFilter" defines methods for accessing properties of "IContextDefinition" instances, according to whose "ILoggedContext" instances are to be filtered.

The interface inherits from the "ICfUnknown" interface.

All methods return CF_SUCCESS after successful execution. In the case of an error, the methods return the corresponding error code.

Members

"GetName" method

Supplies the name of the "IContextDefinition" instance according to whose "ILoggedContext" instances the filtering is performed.

```
CFRESULT GetContextName(CFSTR* p_strName)
```

`p_strName`:

[out]: The name of the "IContextDefinition" instance

"SetName" method

Sets the name of the "IContextDefinition" instance according to whose "ILoggedContext" instances the filtering is performed.

```
CFRESULT SetContextName( CFSTR p_strName)
```

`p_strName`:

[in]: The name of the "IContextDefinition" instance

"GetProviderType" method

The HmiContextProviderType that the "IContextDefinition" instance of the context log entry must have.

```
CFRESULT GetProviderType(CFENUM* p_ProviderType)
```

- `p_ProviderType`:
  [out]: The provider type.
  The enumeration "HmiContextProviderType" can contain the following values:
  - NoContext = 0
  - Calendar = 1
    For "IContextDefinition" instances generated by the PI Option Calendar.
  - PerformanceInsightMachineState = 2
    For "IContextDefinition" instances generated by the PI Option Performance Insight.
  - LineCoordinator = 3
    For "IContextDefinition" instances generated by the PI Option Line Coordinator.
  - UserDefined = 5
    It is a user-defined "IContextDefinition" instance, created by ODK, for example.

"SetProviderType" method

Sets the HmiContextProviderType of an "IContextDefinition" instance for whose "ILoggedContext" instances the filtering is performed.

```
CFRESULT SetProviderType(CFENUM p_ProviderType)
```

- `p_ProviderType`:
  [in]: The provider type.
  The enumeration "HmiContextProviderType" can contain the following values: See "GetProviderType".

"GetOperator" method

Supplies the filter operator.

```
CFRESULT GetOperator(CFSTR* p_Operator)
```

- `p_Operator`:
  [out]: The operator.
  The operator is applied to the value.

"SetOperator" method

Sets the filter operator.

```
CFRESULT SetOperator(CFSTR p_Operator)
```

- p_Operator:
  [in]: The operator
  The operator is applied to the value. The following operators are allowed:
  For values with data type Int and Real:
  - =
  - !=
  - <
  - >
  - <=
  - >=

  For values with data type String:
  - LIKE
  - =

  Strings must always be fully specified.

"GetValue" method

Supplies the "Value" of the "ILoggedContext" instance

```
CFRESULT GetValue(CFVARIANT* p_vtValue)
```

- p_vtValue:
  [out]: The value

"SetValue" method

Sets the "Value" of the "ILoggedContext" instance

```
CFRESULT SetValue(CFVARIANT p_vtValue)
```

- p_vtValue:
  [in]: The value

20.3.10 Reference of the ODK error codes

Reference

Below you will find an overview of the error codes and error descriptions of the ODK interface.

General errors

Error name	Error code	Error description
OPN_E_GENERAL_FAILED	2147496936	General error occurred.
OPN_E_GENERAL_CONNECT_FAILED	2147496967	Connection to runtime failed.
OPN_E_GENERAL_TYPE_NOT_SUPPORTED	2147496985	The type isn't supported.
OPN_E_GENERAL_UNKNOWN_NAME	2147496986	The given name is unknown.
OPN_E_GENERAL_LOADING_FAILED	2147496987	The option can't be loaded.
OPN_E_GENERAL_RESPONSE_TIMEOUT	2147496951	The operation has timed out.
OPN_E_GENERAL_NOT_IMPLEMENTED	2147496954	Error Description Not Availbale
OPN_E_GENERAL_INVALID_SYSTEM	2147498006	Invalid system name.
OPN_E_GENERAL_REQUEST_REJECTED	2147496981	Your request has been rejected.
OPN_E_GENERAL_LICENSE_MISSING	2147496988	There are missing or expired Licenses.
OPN_E_GENERAL_LICENSE_COMPROMISED	2147496989	License: The Module integrity has been compromised.
OPN_E_GENERAL_LICENSE_DISCONNECTED	2147496990	The License is disconnected.

Tag errors

Error name	Error code	Error description
OPN_E_TAG_FAILED	2148545512	General error in Tag model occurred.
OPN_E_TAG_INVALIDARG	2148545513	Invalid arguments passed to function.
OPN_E_TAG_ACCESS_DENIED	2148545532	No access
OPN_E_TAG_INVALIDCREF	2148545514	Invalid CRef found during conversion.
OPN_E_TAG_INVALIDOBJECTSTATE	2148545515	Object not properly initialized.
OPN_E_TAG_NOTSUPPORTED	2148545516	Function not supported.
OPN_E_TAG_OUTOFRANGE	2148545517	The given value is out of range.
OPN_E_TAG_NOTEXISTING	2148545518	The given tag does not exist.
OPN_E_TAG_WRITE_FALS	2148545545	The write operation was partly successful
OPN_E_TAG_READ_FALSE	2148545546	The read operation was partly successful
OPN_E_TAG_DUPLICATE_CREF	2148545549	Duplicate CRef found during conversion.
OPN_E_TAG_INVALID_POINTER	2148545550	Function returned an invalid pointer
OPN_E_TAG_CONVERSION_NOT_ALLOWED	2148545551	Conversion not allowed
OPN_E_TAG_PLC_NOT_CONNECTED	2148545552	PLC not connected
OPN_E_TAG_PLC_PROTOCOL_ERROR	2148545553	Error Description Not Availbale
OPN_E_TAG_INVALID_SUBSCRIPTION	2148545554	Invalid tag subscription
OPN_E_TAG_NO_INIT	2148545519	HmiRuntime not properly initialized.
OPN_E_TAG_UNSUPPORTED_VALUE_TYPE	2148545560	Unsupported value type
OPN_E_TAG_EXISTS	2148545521	The given tag already exists

Error name	Error code	Error description
OPN_E_TAGSET_ALREADY_SUBSCRIBED	2148545610	The TagSet already Subscribed
OPN_E_EMPTY_COLLECTION_SUBSCRIPTION	2148545611	No Tags to Subscribe

Alarm errors

Error name	Error code	Error description
OPN_E_ActiveAlarm_FAILED	2149594088	General error in Alarm model occurred.
OPN_E_ActiveAlarm_INVALID_SUBSCRIPTION	2149594130	Invalid alarm subscription
OPN_E_ActiveAlarm_INVALIDARG	2149594089	Invalid arguments passed to function.
OPN_E_ActiveAlarm_NOINIT	2149594095	HmiRuntime not properly initialized.
OPN_E_ActiveAlarm_INVALIDCREF	2149594090	Invalid CRef found during conversion.
OPN_E_ActiveAlarm_INVALIDOBJECTSTATE	2149594091	Object not properly initialized.
OPN_E_ActiveAlarm_WRONGALARMSTATE	2149594096	The active alarm can't be acknowledged or reset in current state.
OPN_E_ActiveAlarm_INVALID_POINTER	2149594126	Function returned an invalid pointer
OPN_E_ActiveAlarm_NOT_EXISTING	2149594094	The given alarm does not exist.
OPN_E_ActiveAlarm_DUPLICATE_CREF	2149594125	Dublicate CRef found during conversion.
OPN_E_ActiveAlarm_CREATE_FAILED	2149594098	Alarm Create Failed
OPN_E_ActiveAlarm_TIMEOUT	2149594103	Response TimeOut

Plant model errors

Error name	Error code	Error description
OPN_E_CPM_FAILED	2151691240	General error in PlantModel model occurred.
OPN_E_CPM_INVALIDARG	2151691241	Invalid arguments passed to function.
OPN_E_CPM_NODE_NOT_FOUND	2151691275	PlantModel Node Not Exists
OPN_E_CPM_NODEMEMBER_NOT_FOUND	2151691276	PlantModel Node Member Not Exists
OPN_E_CPM_INVALID_POINTER	2151691278	Function returned an invalid pointer
OPN_E_CPM_NO_MEMORY	2151691283	Function returned null pointer
OPN_E_CPM_INVALID_OBJECT_STATE	2151691243	Object not properly initialized.
OPN_E_CPM_UNKNOWN_IDENTIFIER	2151691284	Function returned unknow identifier
OPN_E_CPM_REQUEST_REJECTED	2151691285	Method returned request rejected
OPN_E_CPM_INVALID_CREF	2151691242	Invalid CRef found during conversion.
OPN_E_CPM_INVALID_SUBSCRIPTION	2151691282	Invalid PlantModel node subscription
OPN_E_CPM_NO_INIT	2151691247	HmiRuntime not properly initialized.

Connection errors

Error name	Error code	Error description
OPN_E_CONNECTION_INVALID_PARAM-ETERS	2154837060	Invalid arguments passed to function.
OPN_E_CONNECTION_CONNEC-TION_LOST	2154837061	Connection lost
OPN_E_CONNECTION_INVALID_CREF	2154837062	Invalid connection name passed to func-tion
OPN_E_CONNECTION_INVALID_POINT-ER	2154837063	Invalid pointer
OPN_E_CONNECTION_NOT_EXISTING	2154837064	The connection with the given name does not exist.
OPN_E_CONNECTION_FAILED	2154837065	General error in connection model oc-curred.

Context errors

Error name	Error code	Error description
OPN_E_CONTEXTLOGGING_CRE-ATE_FAILED	2162178060	Create Context Definition failed
OPN_E_CONTEXTLOGGING_TIMEOUT	2162177015	Request TimeOut
OPN_E_CONTEXTLOGGING_INVA-LID_PARAMETERS	2162177001	Invalid Arguments Passed
OPN_E_CONTEXTLOGGING_NOT_SUP-PORTED	2162177004	Not Supported
OPN_E_CONTEXTLOG-GING_START_FAILED	2162178061	Failed to start ContextLogging
OPN_E_CONTEXTLOGGING_FAILED	2162177000	General Error Occurred
OPN_E_CONTEXTLOGGING_NO_INIT	2162177007	Initialization Failed
OPN_E_CONTEXTLOGGING_OB-JECT_STATE	2162177003	Object State is Invalid or CHmiRuntime not properly initialized
OPN_E_CONTEXTLOG-GING_STOP_FAILED	2162178062	Failed to stop ContextLogging
OPN_E_CONTEXTLOG-GING_NODE_NOT_FOUND	2162177035	Plant node not found
OPN_E_CONTEXTLOGGING_DEFINI-TION_NOTEXISTS	2162177006	The given object doesn't exist
OPN_E_CONTEXTLOGGING_DEFINI-TION_EXISTS	2162177009	The given object already exist
OPN_E_CONTEXTLOGGING_SUBSCRIP-TION_EXITS	2162178063	There is already Open Subscription
OPN_E_CONTEXTLOGGING_SUBSCRIP-TION_NOT_OPEN	2162177042	There is no Open Subscription

20.4 WinCC Unified GraphQL

20.4.1 Introduction

WinCC Unified GraphQL API

GraphQL API is a web-based interface for implementing applications (GraphQL clients) that have read and write access to runtime data.

Functional scope

GraphQL enables access to the following runtime data of an HMI device:

	Read	Write	Subscribe to changes
Process values of tags	✓	✓	✓
Active alarms	✓	-	✓

Benefits

Use of Unified GraphQL clients offers the following benefits:

- High user-friendliness, easy integration and few dependencies
- Growing prevalence of GraphQL
- No installation of Unified Runtime on the developer device and when operating the client application
- GraphQL API can be used by all applications that access a network via HTTP and WebSockets (e.g. web applications, desktop applications, console applications).
- You can integrate ready-to-use GraphQL client libraries for a variety of programming languages into your project.
  You can easily extend these libraries through a user-defined GraphQL layer.
- There are ready-to-use GraphQL clients you can use to execute GraphQL operations without having to develop them yourself. An example is the Apollo client used in the quick start section of this user help.
- Built-in API documentation that is available online and always up-to-date
- Precise control of the client over the data processed and transferred by the server

Technical details

- HTTP and WebSockets are the transport layer for client operations and server responses.
- The operations and responses use a JSON-based format.
- The structure of client queries and server responses is described by a schema.
- Each operation specifies a selection set. Only the attributes specified in the selection set are contained in the server response.

20.4.2 Basics

20.4.2.1 Limitations

Unified GraphQL API is subject to the following limitations:

- Access is limited to the runtime to which the GraphQL client is logged in. Access to HMI devices connected via Runtime Collaboration is not possible.
- If a GraphQL client subscribes to two tags and their tag names are renamed in a delta download such that they swap their names, the subscription is closed.
  The GraphQL client must subscribe to the tags again.
- In the existing version of Unified GraphQL, several alarm properties cannot be used as a filter in `activeAlarms` subscriptions and `acitveAlarms` queries.
  For a list of these alarm properties, see section Filtering alarms (Page 8248).

20.4.2.2 Security

This section provides security-relevant information on the GraphQL server.

Secure communication via IIS web server

To ensure secure communication and to protect the GraphQL server and runtime, the GraphQL server is accessed via the IIS web server of Unified Runtime.

Direct access to GraphQL is not supported.

Session duration

A session between the GraphQL server and GraphQL client remains open as long as there is activity, e.g. a regularly executed query.

All sessions of all users are closed in the following cases:

- Start of the GraphQL server with a different manager number
- Deletion of the internal session database

Closing all sessions of a user

You have the following options:

- The user logs out from one of his sessions and, in so doing, passes the value `True` for parameter `allSessions`.
- The administrator changes the password of the user in the engineering system. The administrator logs in to GraphQL with the changed login data of the user. The administrator logs the user out and, in so doing, passes the value `True` for parameter `allSessions`.

20.4.2.3 More information

You can find more information on GraphQL here:

- About GraphQL in general: https://graphql.org/ (https://graphql.org/)
- About Apollo Studio: https://www.apollographql.com/ (https://www.apollographql.com/)
- Solutions to technical problems concerning GraphQL: https://stackoverflow.com/ (https://stackoverflow.com/)

20.4.3 Quick start

20.4.3.1 Purpose of this quick start

This quick start demonstrates, using a ready-to-use GraphQL client freely available in Apollo Studio, how to establish a connection to a Unified GraphQL server, log in to runtime and then query, manipulate and subscribe to runtime data.

The quick start sets you up for subsequent experimentation with the GraphQL client of Apollo and helps you to get started working with WinCC Unified GraphQL.

20.4.3.2 Requirements

To follow the instructions and examples of the quick start, the following general requirements must be met:

- WinCC Unified Runtime is installed on a PC.
- A runtime project is running on the PC.
- The GraphQL Server Manager on this PC is effective via IIS through: **https://localhost/graphql/**

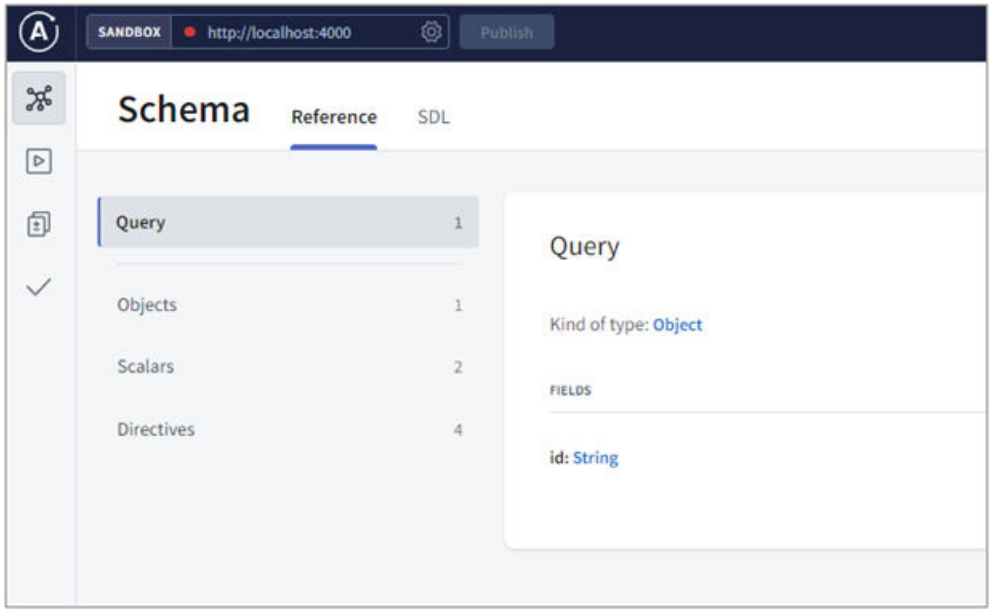
See also

Setting up the GraphQL client (Page 8224)

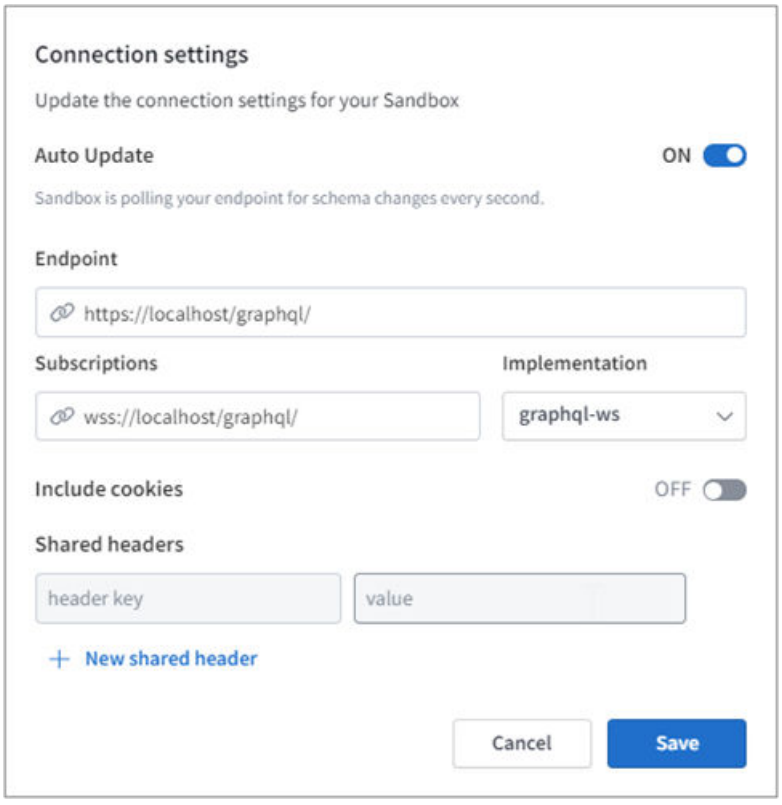
20.4.3.3 Setting up the GraphQL client

Procedure

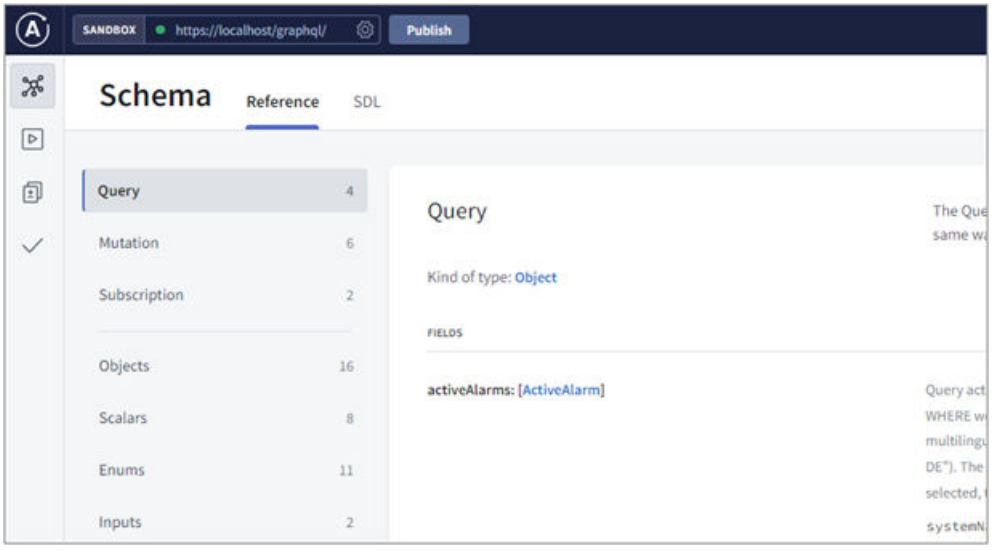
1. Start a web browser on a device that has access to the runtime server.
2. Enter the following address in the address line and press Enter:
   https://studio.apollographql.com/sandbox/explorer (https://studio.apollographql.com/sandbox/explorer)
   Apollo Studio opens:



3. Click on the gear icon in the address line.
   The configuration settings for the GraphQL client open. Apply the following settings:



Schema introspection is active for Unified GraphQL Server. That is why you see a green dot in the address line after configuration of the GraphQL client. In addition, the documentation of the GraphQL API is available in the client:



See also

Requirements (Page 8223)

Basics on the schema (Page 8234)

20.4.3.4 Logging in to the GraphQL server

Note

For information on the SWAC login, see section "loginSWAC" mutation (Page 8251).

In order for a GraphQL client to access runtime data, it must be logged in to Runtime with a user.

Requirement

In addition to the general requirements, the following is required:

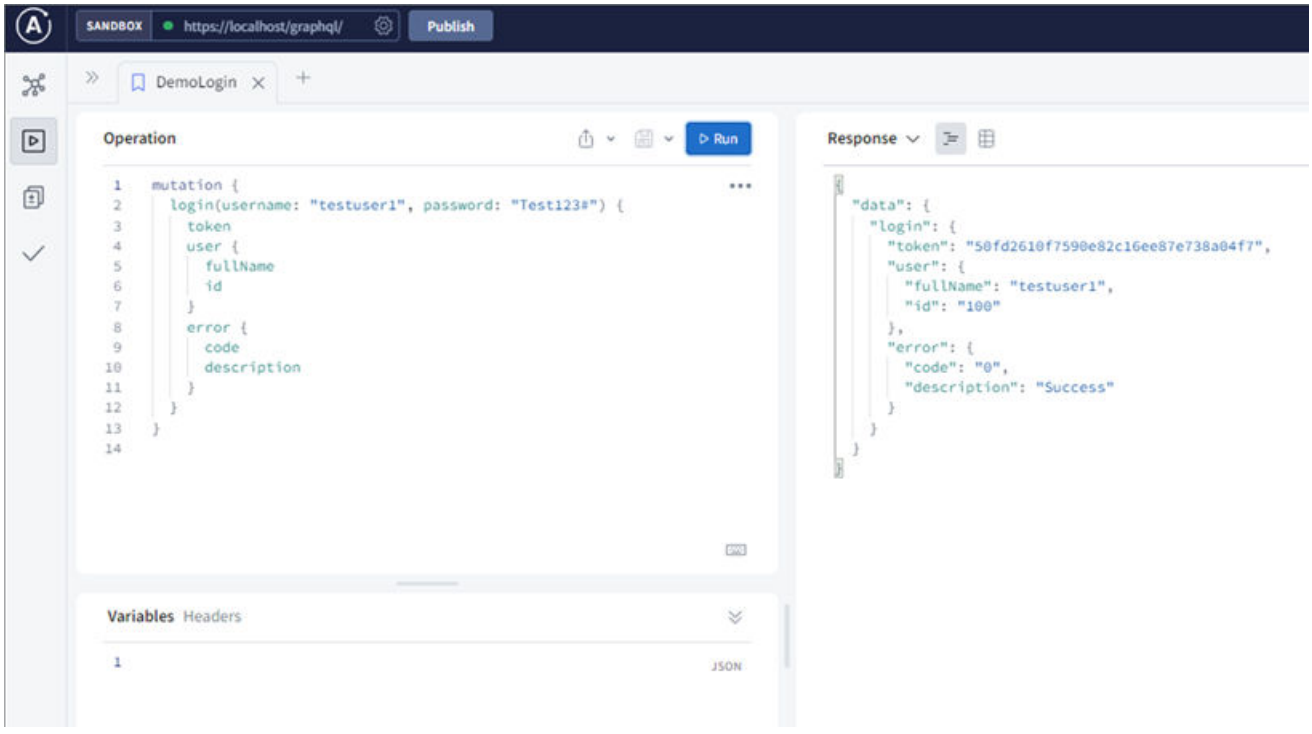
- The GraphQL server has been set up.

Procedure

1. Log the GraphQL client in to the GraphQL server by calling the mutation `login`, as shown below.
   Pass the login data of a UMC user configured for runtime to the operation as input parameter.

Result

The server opens a session. The server response contains the ID of the authorization token:



You use this token for subsequent requests of GraphQL operations. If the client has been inactive for a while, the token expires. The client must log in again.

Note

By requesting the `extendSession`mutation, you keep the session open in spite of inactivity.

See also

Requirements (Page 8223)

Setting up the GraphQL client (Page 8224)

Authorizing an operation request (Page 8231)

"login" mutation (Page 8250)

"extendSession" mutation (Page 8252)

20.4.3.5 Executing a GraphQL operation

Requirement

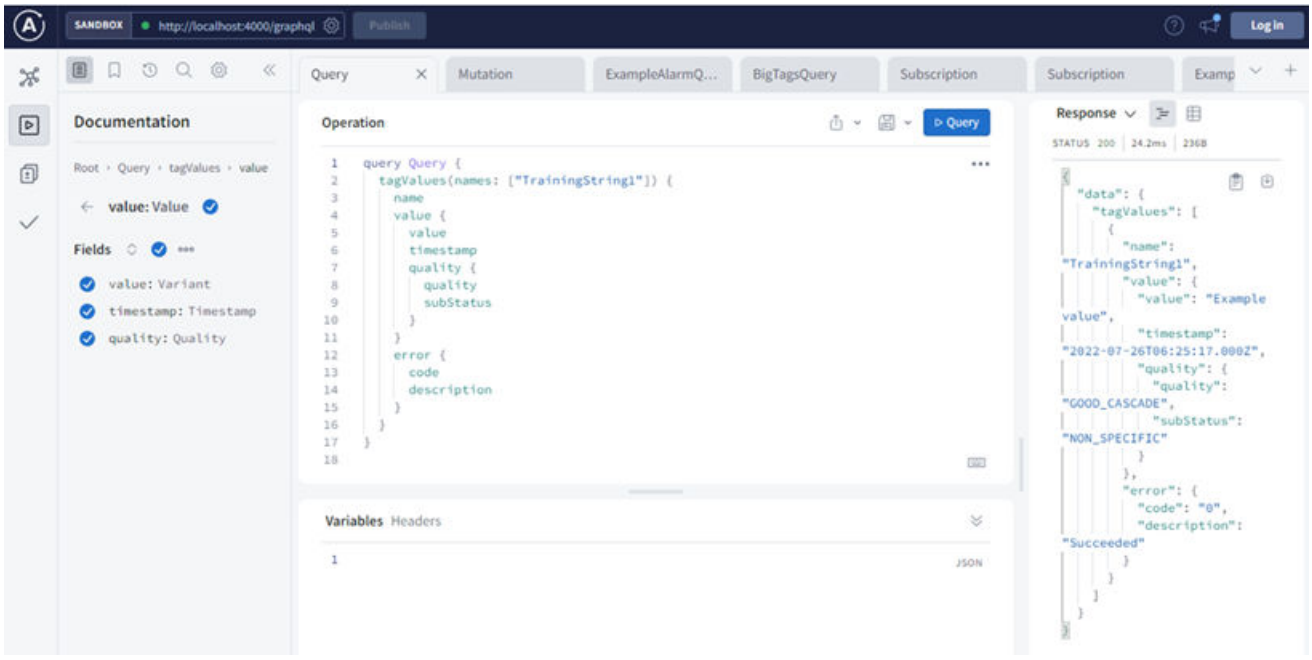
In addition to the general requirements, the following is required:

- The GraphQL server has been set up.
- The GraphQL client is logged in to the server. The authorization token issued at login is still valid.
- The user with whom the GraphQL client is logged in has the following function rights:
  - Query or subscription: "GraphQL - read access" or "GraphQL - read/write access"
  - Mutation: "GraphQL - read/write access"

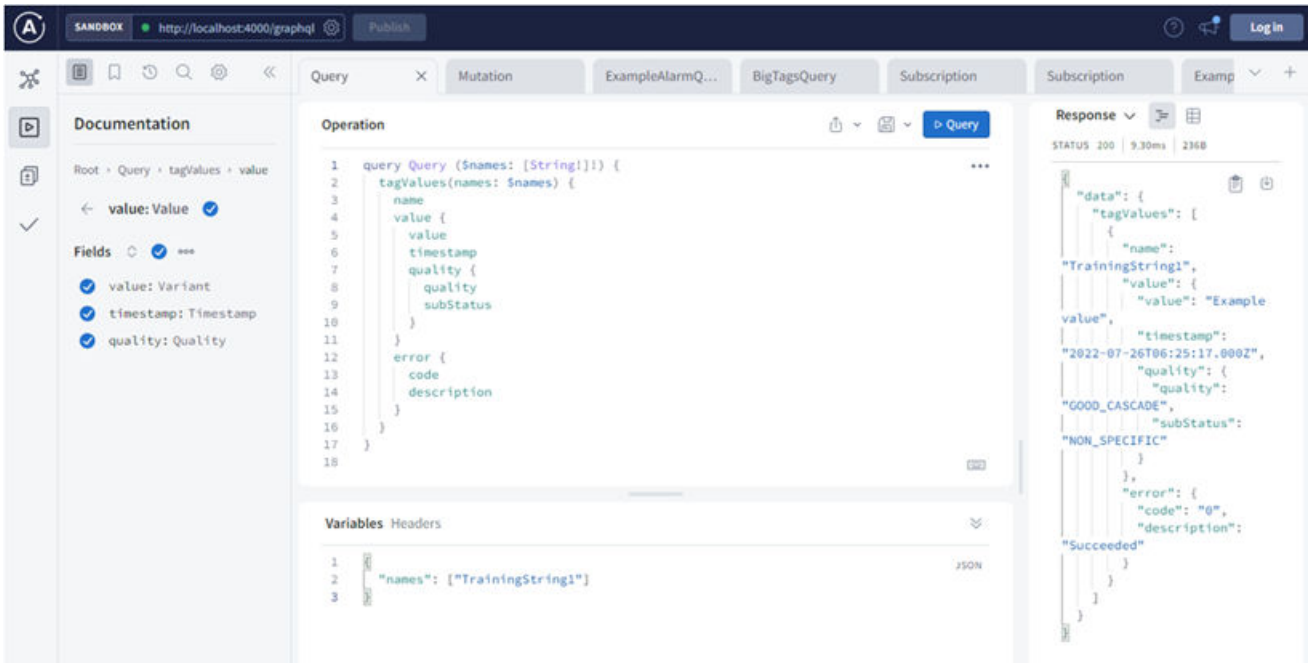
Procedure

To execute GraphQL operations for a Unified GraphQL client, proceed as described below. The basic procedure is the same for all three operation types.

1. In Apollo Studio, select the Explorer view and input the desired operation in the "Operation" panel.
   You have the following options:
   - Manually entering the operation request in the panel
   - Pointing the mouse to the desired position in the panel, changing to the Documentation view, selecting the desired operation there and adding it with the "+" icon
2. Specify the input parameters.
   You have the following options:
   - Manually passing the parameters in the operation request:



– Creating a tag in the "Variables" panel and using it in the "Operation" panel. For this you enter the character "$" followed by the tag name in the "Operation" panel:



Note

Tag as input parameter

Ensure that the tag exists in the connected runtime and that the tag name is written correctly.

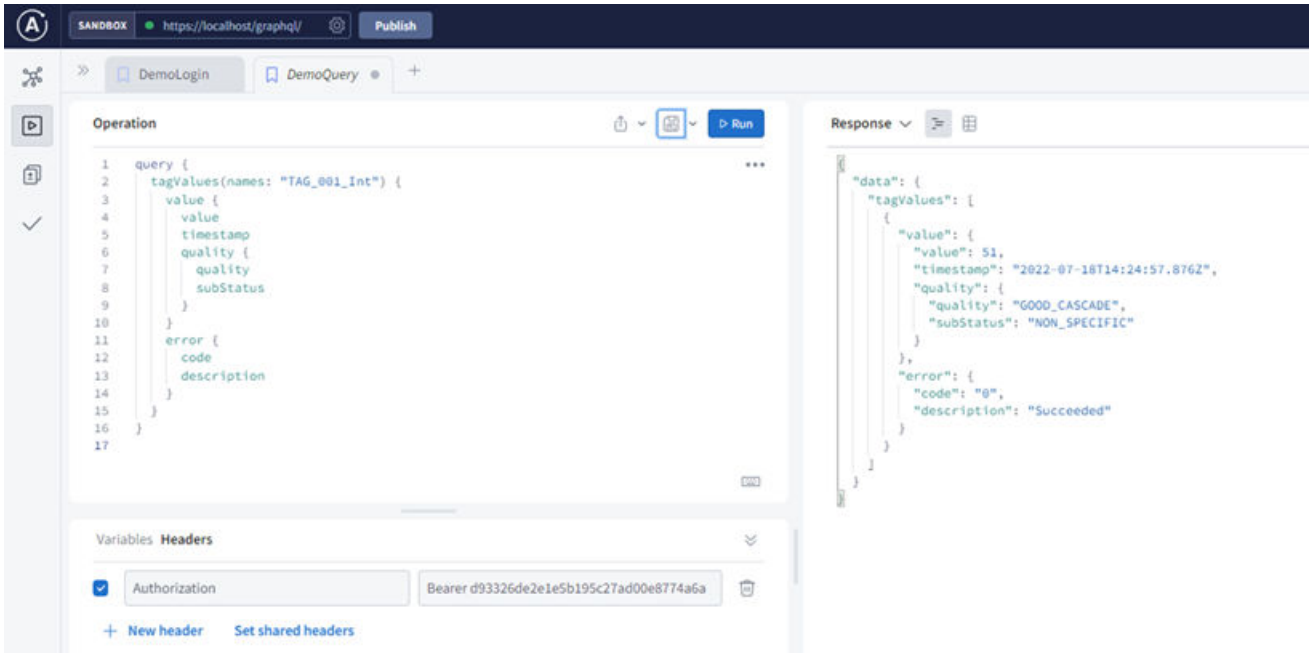
3. In the "Headers" panel, define additional HTTP header values, e.g. obligatory authorization tokens for access to tags and active alarms.
4. Click the button for executing the operation.

Result

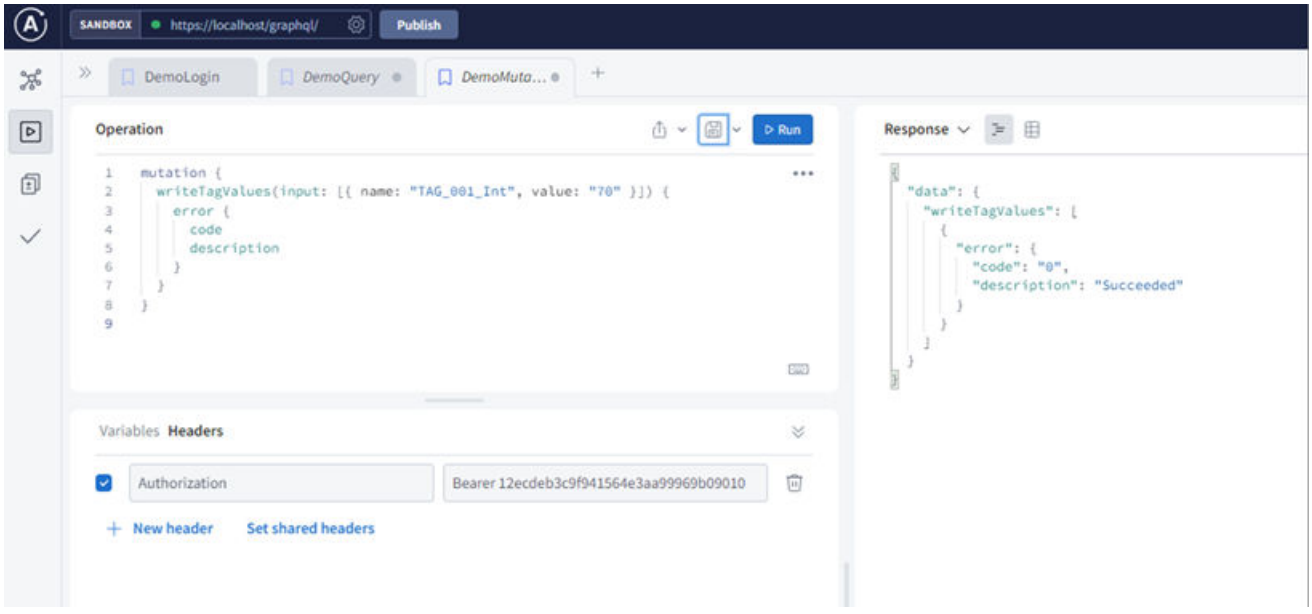
- The operation is executed.
- If the client is logged in, you see the response of the GraphQL server in the "Response" panel. If the client is not logged in, you see a corresponding error message in the "Response" panel.

Examples for the various operation types

- Query type:

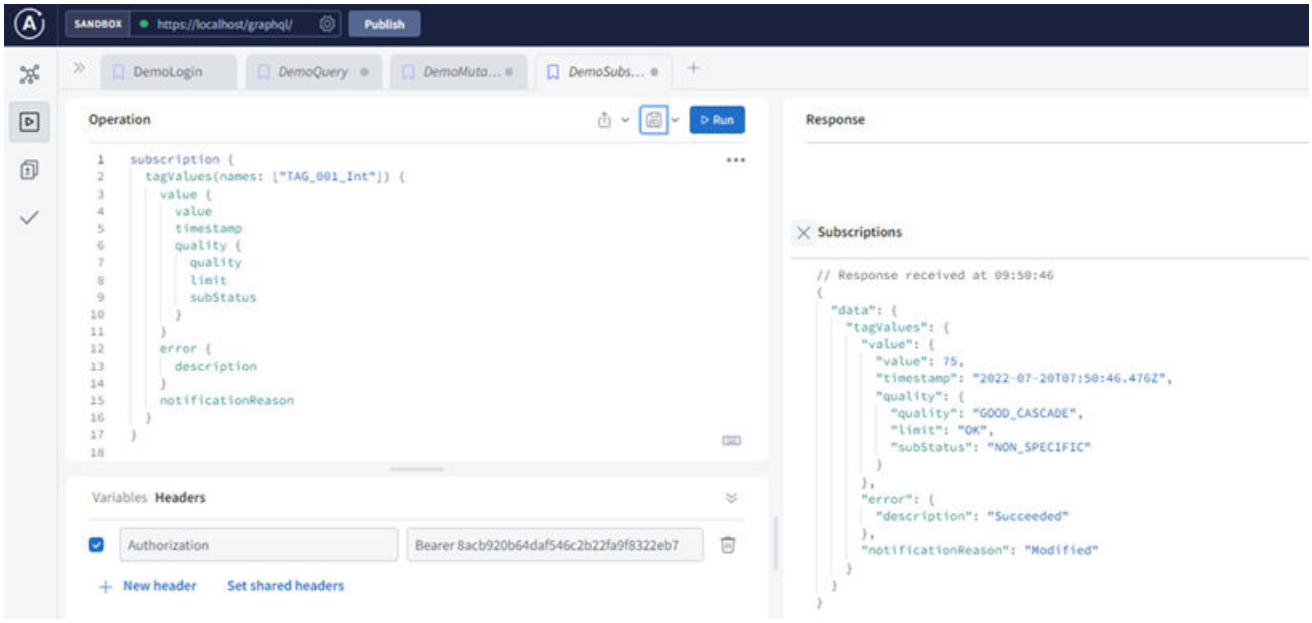


- Mutation type:



If you execute a query for the tag after the mutation, you receive the modified value.

- Subscription type:



If you execute a mutation for the tag after the subscription, you receive a notification.

Closing a subscription

To close a subscription, click "X" button in the "Subscriptions" panel.

Note

Subscriptions in Apollo Studio

There is only one active subscription in the GraphQL client in Apollo Studio. If you start a second subscription, it replaces the first one. The second subscription is then the active subscription.

See also

Requirements (Page 8223)

Authorizing an operation request (Page 8231)

20.4.3.6 Authorizing an operation request

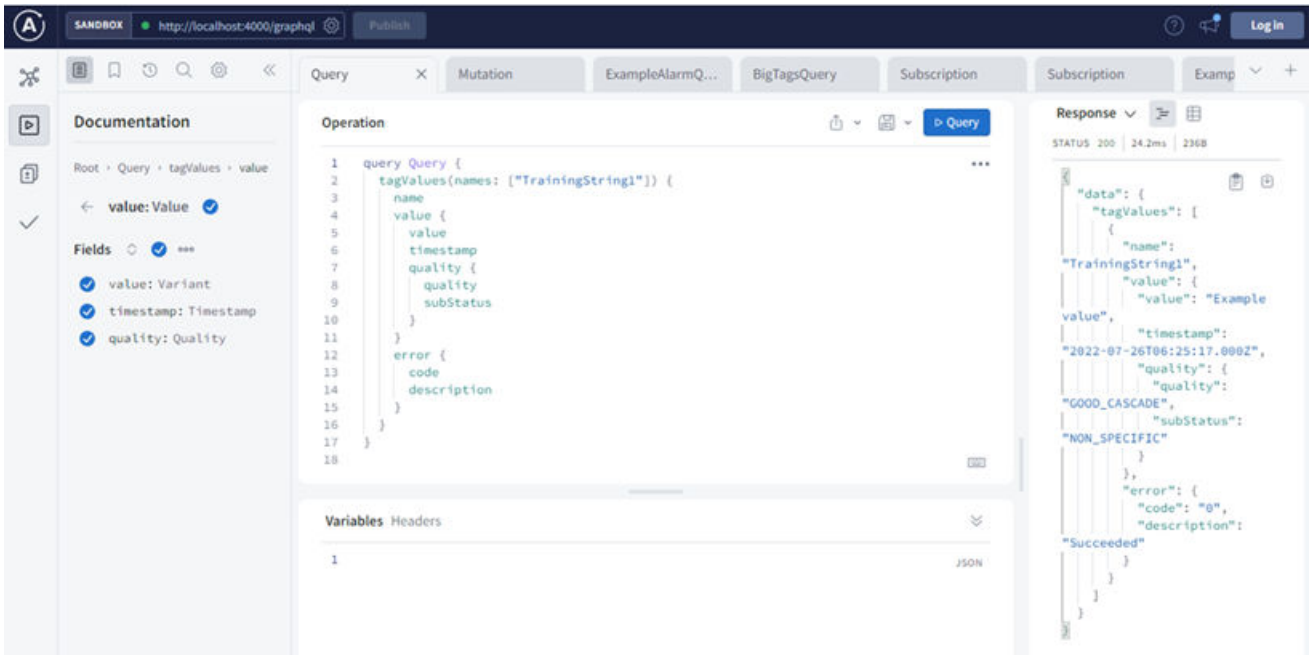
Requirement

In addition to the general requirements, the following is required:

- The GraphQL server has been set up.
- The GraphQL client is logged in to the server. The authorization token issued at login is still valid.
- The desired operation is displayed in the "Operation" panel.

Procedure

1. Select the "Headers" panel of the operation.
2. Add a header if necessary.
3. Enter the following header data:
   - Key: "Authorization"
   - Value: "Bearer <Token ID from the login response>"
4. Activate the header.



See also

Requirements (Page 8223)

Executing a GraphQL operation (Page 8227)

Logging in to the GraphQL server (Page 8226)

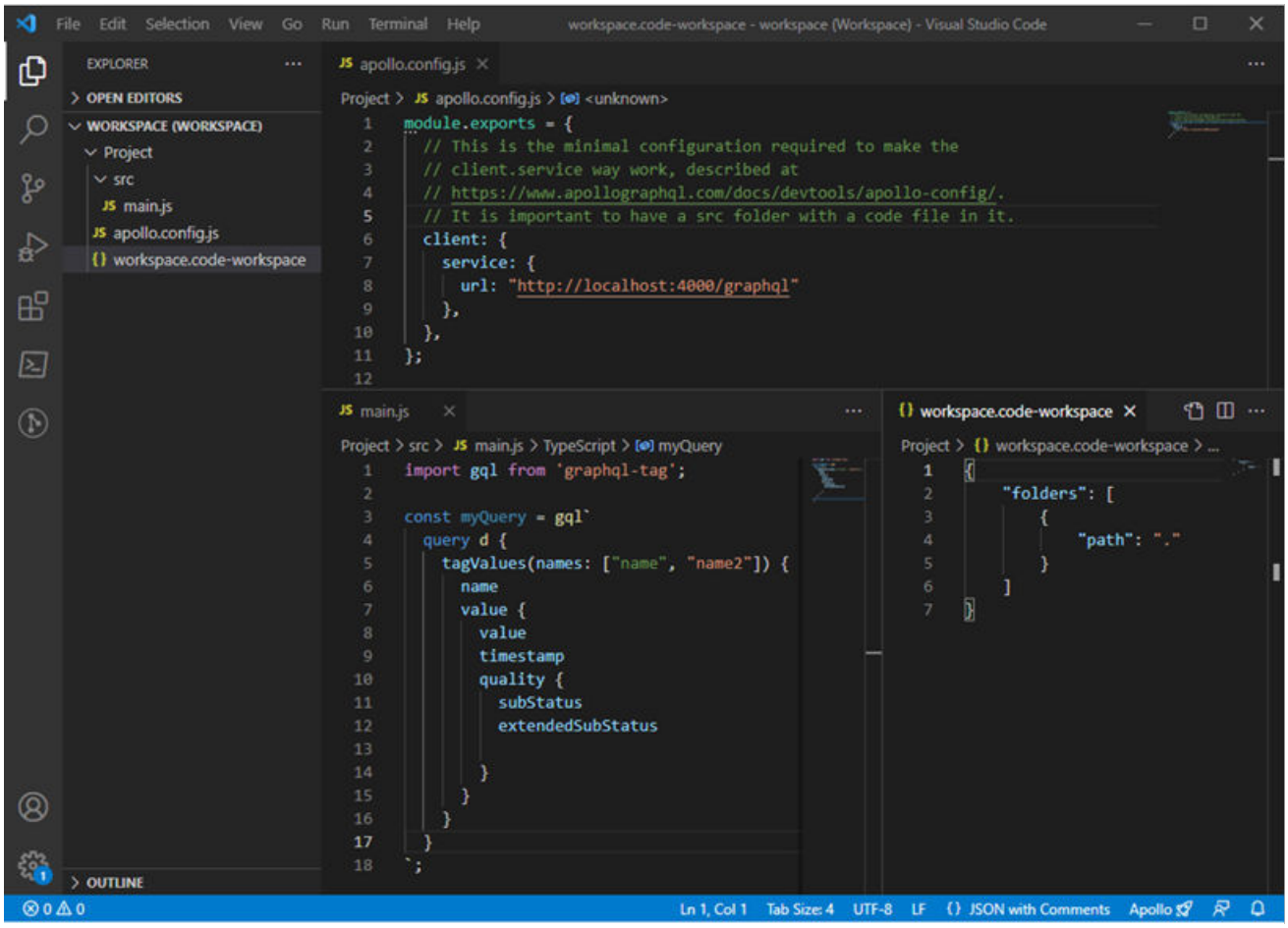
20.4.3.7 Using the syntax highlighting and autocompletion functions of Apollo

Introduction

This section describes how to integrate the Apollo GraphQL extension for syntax highlighting and autocompletion into Visual Studio Code.

Procedure

1. Download the Apollo GraphQL extension from the Visual Studio Marketplace (https://marketplace.visualstudio.com/items?itemName=apollographql.vscode-apollo).
2. Install it.
3. Apply the following settings:



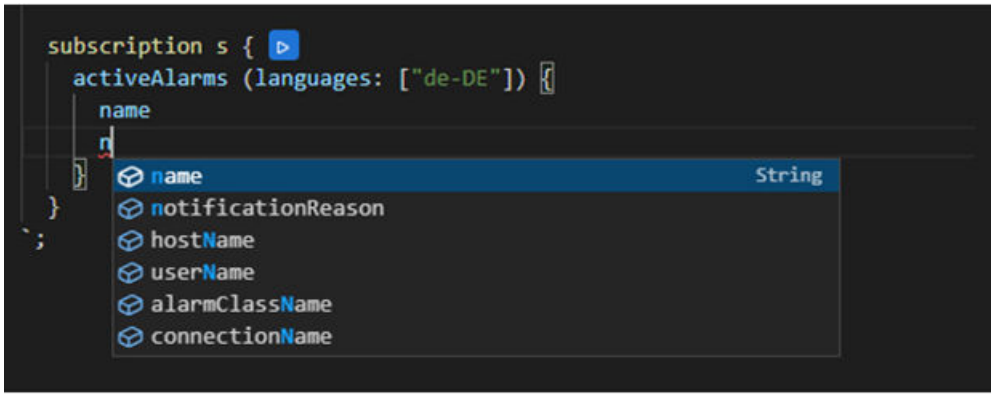
Result

When the schema inspection of the GraphQL server is active and the GraphQL server is running, Visual Studio supports you with additional features.

Example:

- Hover the mouse pointer over a GraphQL code. A tooltip with additional information opens:

- When entering GraphQL code, you receive autocomplete suggestions:



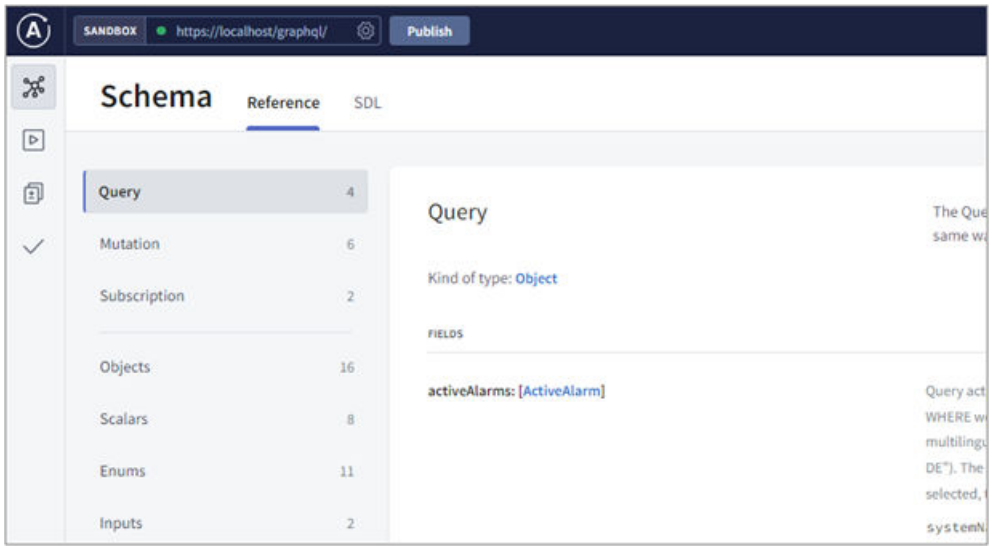
20.4.4 Schema

20.4.4.1 Basics on the schema

The GraphQL schema is an object in JSON format that describes the GraphQL API, which is understandable by humans and applications alike.

If the schema of the GraphQL server is known to the GraphQL client, the client supports you with autocompletion and syntax checking.

In Apollo Studio, you can see that the client knows the schema by the green dot in the address bar. The documentation of the GraphQL API is then available in the client:



Automatically querying the schema

The schema inspection is active for GraphQL server by default. Therefore, GraphQL clients automatically query the schema from the server.

Manually downloading the schema

You can also download the schema manually, e.g. by executing the following npx command in the Windows command line:

```
npx apollo-cli download-schema http://localhost:4000/graphql --
output schema.json
```

More information on the manual download of the schema can be found here (https://stackoverflow.com/questions/37397886/get-graphql-whole-schema-query).

Syntax highlighting and autocompletion in editors

To make use of syntax highlighting and autocompletion in editors when implementing a GraphQL client, extensions suitable for the development environment must be installed. For example, you can install the Apollo GraphQL extension for Visual Studio Code.

See also Using the syntax highlighting and autocompletion functions of Apollo (Page 8232).

20.4.4.2 Structure of a client query

Overview

```
1 query
2 {
3    tagValues(names: "DEMO_CPU_TEMPERATURE")
4    {
5       value
```

```
6       {
7          value
8          timestamp
9          quality
10          {
11             quality
12          }
13       }
14       error
15       {
16          code
17          description
18       }
19    }
20 }
```

Line	Description
1	The operation type of the requested operation, in this example: `query`
3	Name of the requested operation and list of input parameters in the following format: (<Name_Parameter1>: "<Value>", <Name_Parameter2>: "<Value>")
4 to 19	The selection set with the attributes requested from the server

Structure of the selection set

The selection set is where you specify for the operation request which attributes the server returns. You can request all the attributes defined in the return type of the operation.

Notation:
`Operation name ( <Input parameter>) {<Selection set> }`

Where <Selection set> stands for a comma-separated list of the attribute names requested by the server.

Specification of type-based attributes:
<Name of type-based attribute> { <Comma-separated list of attribute names> }

Example of type-based attributes: The attributes `value`, `quality` and `error` from the selection set above.

20.4.5 Reference of GraphQL API

20.4.5.1 General information on GraphQL API

An important feature of GraphQL API is its high user-friendliness. For Unified GraphQL API, this means:

- Tags and active alarms from runtime are addressed using their name and not their ID. The names are configured in the engineering.
- QualityCodes are represented by the string assigned to them in an enumeration. This improves readability of the code.

- Color values are translated to the standard RGBA notation.
- Date information is represented according to ISO 8601 (https://en.wikipedia.org/wiki/ISO_8601).
- To execute bulk operations, it is possible to pass GraphQL operations as input parameter lists. The operation is then applied to each list element.
- Internal error messages are translated into user-level messages.

20.4.5.2 GraphQL operation types

Overview

GraphQL operations have one of the following operation types:

- Query
- Mutation
- Subscription

Queries

You use queries for read operations. The query sends an HTTPS request to the GraphQL server.

Example: Querying tag values or alarms that satisfy a filter criterion

Mutations

You use mutations for write operations. The mutation sends a one-time query to the GraphQL server.

Example: Writing the value and QualityCode of the value to a tag

Subscriptions

You use subscriptions so that your GraphQL client can react to runtime events promptly without having to continually query the runtime status. The subscription creates a permanent WebSocket connection between the GraphQL server and client. When a client subscribes to an alarm or tag, the server notifies the client when changes to the alarm or tag are active.

Example: The GraphQL client is to execute a certain operation when the value of a tag changes.

Subscriptions are usually closed by the client, but they can also be closed by the server or mutually by both sides, e.g. in the event of a network failure.

Subscriptions use the graphql-transport-ws protocol. Configure this protocol for your GraphQL library or implement WebSocket communication based on this protocol.

Client-side GraphQL libraries provide different options for closing subscriptions, depending on the programming language being used. For example, Python uses an asynchronous iterator that the notification runs through. When the loop is exited, the subscription is closed.

See also

Disconnection by server (Page 8266)

20.4.5.3 Operations for tags

"tagValues" query

Requirements

- The user who is logged in to runtime for the GraphQL client has the rights "GraphQL - read access" or "GraphQL - read/write access" in runtime.

Description

```
tagValues(names: [String]): [TagValueResult]
```

Operation name	`tagValues`
Operation type	`query`
Function	Reads the tags specified in input parameter `names`. Read access is possible to tags with a simple data type, as well as to elements of structure tags and arrays with a simple data type.
Input parameters	`names` • Mandatory parameter • Data type: String • List of names of the tags whose properties you want to read. Notation: `names:["<Name Tag1>", "<Name Tag2>", "<...>"]` Example: `names:["motor1.tempMax", "motor1.tempMin"]` Addressing of tags: – Addressing a tag with simple data type: `<Tag name>` – Addressing an element of a structure tag: `<Structure tag name>.<Element name>` Example: `motor1.speed` – Addressing an individual element of an array: `<Array name>[<Index>]` Example: `FloatArrayTag[0]` – Address all tags of an array: `<Array name>` Example: `FloatArrayTag` The tags are returned in an array in the response.
Selection set	Mandatory specification Specify which attributes the server returns for the queried tags. You can request the attributes defined in the `TagValueResult` type. For information on the notation, see section Structure of a client query (Page 8235).

Server response	Supplies the attributes requested in the selection set for all tags specified with `names` as key-value pairs of a JSON data record.
Error messages (`code: description`)	• `0: Success` • `2: Cannot resolve provided name` • `202: Only leaf elements of a Structure Tag can be addressed`

Example

```
queryExampleTagQuery{
  tagValues(names:["ExampleTagName1", "ExampleTagName2"]){
    name
    value{
      value
      timestamp
      quality
      {
        quality
        subStatus
      }
    }
    error{
      description
      code
    }
  }
}
```

See also

"TagValueResult" type (Page 8261)

"writeTags" mutation

Requirements

- The user who is logged in to runtime for the GraphQL client has the right "GraphQL - read/write access" in runtime.

Description

```
writeTagValues(
      input: [TagValueInput],
      timestamp: Timestamp
      quality: QualityInput
    ): [WriteTagValuesResult]
```

Operation name	`writeTagValues`
Operation type	`mutation`
Function	Writes the tags specified in input parameter `input`. Write access is possible to tags with a simple data type, as well as to elements of structure tags and arrays with a simple data type. The mutation allows you to write multiple tags or multiple properties of the same tag with a single operation request.

Input parameters	`input`
	• Mandatory parameter
	• Data type: String
	• A list of the tags and values that you want to write
	Notation: `input:[ {<Information on tag 1>}, {<Information on tag 2>},{<...>}]`
	Example: `input: [{name: "motor1.speed", value: "100"}, {name: "motor2.speed", value: "140"}]`
	Make the following entries for each tag:
	– `name`: For addressing the tag
	Addressing a tag with simple data type: `<Tag name>`
	Addressing an element of a structure tag: `<Structure tag name>.<Element name>`
	Example: `motor1.speed`
	Addressing an individual element of an array: `<Array name>[<Index>]`
	Example: `FloatArrayTag[0]`
	It is not possible to address all elements of an array together.
	– `value`: The process value that is written to the tag
	The data type of the passed value must be convertible to the data type of the tag.
	– (Optional) `timestamp`: The time stamp of the process value as `Timestamp`[1]
	– (Optional) `quality`: The QualityCode of the process value as `QualityInput`
	`timestamp`
	• Optional
	Written to all tags from `input` for which no time stamp has been passed.[1]
	If the input parameter is empty, the current system time is used as the time stamp.
	• Data type: `Timestamp`
	`quality`
	• Optional
	Written to all tags from `input` for which no QualityCode has been passed.
	If the input parameter is empty, "GOOD" is used as QualityCode.
	• Data type: `QualityInput`

Selection set	Mandatory specification Specify which attributes the server returns for the written tags. You can request the attributes defined in the `WriteTagValueResult` type. For information on the notation, see section Structure of a client query (Page 8235).
Server response	Supplies the attributes requested in the selection set for all tags specified with `input` as key-value pairs of a JSON data record.
Error messages (`code: description`)	• `0: Success` • `2: Cannot resolve provided name` • `201: Cannot convert provided value to data type` • `202: Only leaf elements of a Structure Tag can be addressed`

[1] The timestamp must be newer that the current timestamp of the tag and must not be in the future.

Example

```
mutation ExampleTagValueWrite {
  writeTagValues(input:[
    {
      name:"ExampleStringTag",
      value: "Example value text”
    },
    {
      name:"ExampleIntegerTag",
      value: 42,
      timestamp:"2022-07-26T06:25:15Z"
    }
  ], quality:{quality: GOOD_NON_CASCADE}){
    name
    error{
      code
      description
    }
  }
}
```

See also

"WriteTagValueResult" type (Page 8263)

"QualityInput" type (Page 8259)

"tagValues" subscription

Requirements

- The user who is logged in to runtime for the GraphQL client has the rights "GraphQL - read access" or "GraphQL - read/write access" in runtime.

Description

```
tagValues(names: [String]): TagValueNotification
```

Operation name	`tagValues`
Operation type	subscription
Function	Subscribes the tags specified in input parameter `names`.
Input parameters	`names` • Mandatory parameter • Data type: String • List of names of the tags whose properties you want to subscribe to. Notation: `names:["<Name Tag1>", "<Name Tag2>", "<...>"]` Addressing of tags: – Address an element of a structure tag: `<Name of the structure tag>.<Name of the element>` – Address a single element of an array: `<Name of the array>[<Index>]` Example: `FloatArrayTag[0]` – Address all tags of an array: `<Name of the array>` Example: `FloatArrayTag` The tags are returned in an array in the response.
Selection set	Mandatory specification Specify which attributes the server returns for the subscribed tags. You can request the attributes defined in the `TagValuesNotification` type. For information on the notation, see section Structure of a client query (Page 8235). Recommendation: Request `name` and `notifcationReason`.
Server response	Until the subscription is closed, supplies an initial notification for the subscribed tags with the current values of the attributes requested in the selection set as key-value pairs of a JSON data record, and supplies subsequent notifications when a subscribed tag changes.
Error messages (`code: description`)	• `0: Success` • `2: Cannot resolve provided name` • `202: Only leaf elements of a Structure Tag can be addressed`

Closing a subscription

The subscription of a tag closes implicitly:

- When the tag is renamed.
- When the tag is deleted.

When you close the subscription in the GraphQL client explicitly, all previously subscribed tags are no longer subscribed.

Example

```
subscriptionSubscription{
  tagValues(names:["Example_Tag_1", "Example_Tag_2"]){
    name
    value{
      value
      timestamp
      quality{
        quality
        substatus
      }
    }
    error{
      code
      description
    }
    notificationReason
  }
}
```

See also

"TagValueNotification" type (Page 8260)

20.4.5.4 Operations for alarms

"activeAlarms" query

Requirements

- The user who is logged in to runtime for the GraphQL client has the rights "GraphQL - read access" or "GraphQL - read/write access" in runtime.

Description

```
activeAlarms(
      systemNames: [String]
      filterString: String
      filterLanguage: String
```

```
    languages: [String]
  ): [ActiveAlarm]
```

Operation name	`activeAlarms`
Operation type	`query`
Function	Queries the active alarms from the server.

Input parameters	`systemNames`
	• Optional
	• Data type: String
	• Reserved for future versions.
	• Specifies the runtime whose alarms are queried.
	In the current version, the queried alarms are always the alarms of the runtime connected to the GraphQL server.
	`filterString`
	• Optional
	• Data type: String
	• Defines filter criteria for the alarms. See section Filtering alarms (Page 8248).
	If you do not define a filter, the server returns all alarms active in runtime.
	`filterLanguage`
	• Optional
	• Data type: String
	• When `filterString` uses a string with a translatable alarm property, you use `filterLanguage` to define which language is used for the comparison.[1]
	• Default value: "en-US"
	`languages`
	• Optional
	• Data type: String array
	• Defines the languages in which the server returns the texts of translatable alarm properties and the order of the requested languages.[1]
	Example:
	`queryLanguageExample { activeAlarms(languages:["en-US", "de-DE"] { name, eventText, alarmText1, alarmClassName, state, raiseTime, priority } }`
	The server response supplies the attributes specified in the selection set for the queried alarms. Translatable properties, such as `eventText`, are each returned in a string array. The element with index 0 always contains the English text and the element with index 1 the German text.
	• Default value: "en-US"

Selection set	Mandatory specification Specify which attributes the server returns for the queried alarms. You can request the attributes defined in the `ActiveAlarm` type. For information on the notation, see section Structure of a client query (Page 8235).

Server response	Supplies the attributes requested in the selection set for the queried alarms as key-value pairs of a JSON data record.
Error messages (`code` and `description`)	• `0: Success` • `301: Syntax error in query string` • `302: At least one of the requested languages is invalid` • `303: The provided filter language is invalid`

[1] Languages must be specified in ISO language code format (e.g. "en-US", "de-DE"). Note that the entry is case-sensitive.

Example

```
queryExampleAlarmQuery
{
   activeAlarms(languages:["en-US", "de-DE"],
     filterLanguage:"en-US",
     filterString:(raiseTime <= '2022-07-26T09:00' AND userName
LIKE '* Doe') OR priority > 2")
     {
         name
         eventText
         alarmText1
         alarmClassName
         state
         raiseTime
         priority
     }
}
```

See also

"ActiveAlarm" and "ActiveAlarmNotification" types (Page 8254)

"Error" type (Page 8258)

"activeAlarms" subscription

Requirements

- The user who is logged in to runtime for the GraphQL client has the rights "GraphQL - read access" or "GraphQL - read/write access" in runtime.

Description

```
activeAlarms(
      systemNames: [String]
      filterString: String
      filterLanguage: String
      languages: [String]
    ): ActiveAlarmNotification
```

Operation name	`activeAlarms`
Operation type	`subscription`
Function	Subscribes to active alarms.

Input parameters	`systemNames`
	• Optional
	• Data type: String
	• Reserved for future versions.
	• Specifies the runtime whose alarms are subscribed to.
	In the current version, the queried alarms are always the alarms of the runtime connected to the GraphQL server.
	`filterString`
	• Optional
	• Data type: String
	• Defines filter criteria for the alarms. See section Filtering alarms (Page 8248).
	If you do not define a filter, you subscribe to all alarms active in runtime.
	`filterLanguage`
	• Optional
	• Data type: String
	• When `filterString` uses a string with a translatable alarm property, you use `filterLanguage` to define which language is used for the comparison.[1]
	• Default value: "en-US"
	`languages`
	• Optional
	• Data type: String array
	• Defines the languages in which the server returns the texts of translatable alarm properties and the order of the requested languages.[1]
	Example:
	`queryLanguageExample { activeAlarms(languages:["en-US", "de-DE"] { name, eventText, alarmText1, alarmClassName, state, raiseTime, priority } }`
	The initial notification and the subsequent notifications supply the attributes specified in the selection set for the subscribed alarm. Translatable properties, such as `eventText`, are each returned in a string array. The element with index 0 always contains the English text and the element with index 1 the German text.

Selection set	Mandatory specification Specify which attributes the server returns for the subscribed alarms. You can request the attributes defined in the `ActiveAlarmNotification` type. For information on the notation, see section Structure of a client query (Page 8235). Recommendation: Request `name` and `notifcationReason`.

Server response	Until the subscription is closed, the server sends an initial notification and sends subsequent notifications when an attribute of one of the subscribed alarms specified in the selection set changes. The notifications contain the current values of the attributes requested in the selection set as key-value pairs of a JSON data record.
Error messages (`code` and `description`)	• `0: Success` • `301: Syntax error in query string` • `302: At least one of the requested languages is invalid` • `303: The provided filter language is invalid`

[1] Languages must be specified in ISO language code format (e.g. "en-US", "de-DE"). Note that the entry is case-sensitive.

Example

```
queryExampleAlarmQuery
{
   activeAlarms(languages:["en-US", "de-DE"],
     filterLanguage:"en-US",
     filterString:(raiseTime <= '2022-07-26T09:00' AND userName
LIKE '* Doe') OR priority > 2")
     {
         name
         notificationReason
         eventText
         alarmText1
         alarmClassName
         state
         raiseTime
         priority
     }
}
```

See also

"Error" type (Page 8258)

"ActiveAlarm" and "ActiveAlarmNotification" types (Page 8254)

Filtering alarms

Introduction

You use input parameters of `activeAlarms` queries and `activeAlarms` subscriptions to define filters that control which alarms are read or subscribed to. This improves the performance of your GraphQL clients.

Limitation

In the current version of Unified GraphQL, you cannot filter by the following alarm properties:

- `state`
- `stateMachine`
- `changeReason`
- `sourceType`
- `suppressionState`
- `invalidFlags`
- `producer`
- `userResponse`
- `textColor`
- `backColor`
- `valueQuality`
- `quality`

Input parameter "filterString"

You use input parameter `filterString` to define the filter criteria. The filter string corresponds to the WHERE part of a valid ChromQueryLanguage string with addition of the keyword WHERE.

Note

ChromQueryLanguage (QCL) is based on SQL. You can also use most of the valid, simple SQL expressions in CQL.

Wildcards permitted in `filterString`:

*	Replaces any number of characters.
?	Replaces exactly 1 character.

Operators permitted in `filterString`:

>
>
=

LIKE	
()	For grouping of expressions
AND	For logically combining expressions
OR	

Note

Filtering by the "path" property

`path` is the full name of the alarm. It has the following components:

- Alarm of a tag: <System name>::<Tag name>:<Alarm name>
- Alarm of an element of a structure tag or array: <System name>::<Tag name>.<Element path>:<Alarm name>
  Delimiter for the components of the element path: "."

If the GraphQL client only has access to one runtime system or you want to query the alarms of all connected systems, you can replace the system name with the wildcard "*".

Input parameter "filterLanguage"

If `filterString` uses a translatable alarm property, use input parameter `filterLanguage` to specify the language of the text.

Specify the language in ISO language code format (e.g. "en-US", "de-DE"). Note that the entry is case-sensitive.

Examples

- `filterString: "alarmClassName = 'SystemNotification'"`
  Returns all active alarms with alarm class "SystemNotification".
- `filterString: "(raiseTime <= '2022-07-26T09:00' AND userName LIKE '* Doe') OR priority > 2"`
  Returns all alarms that were active before 2022-07-26 9:00 and whose associated user has the last name "Doe" or whose priority is greater than 2.
- `filterString: "eventText LIKE 'Maximum value exceeded'"` and `filterLanguage: "en-US"`
  The response returns only alarms that have the value "`Maximum value exceeded`" for `eventText` in English.

  Note

  You specify the languages and the order in which the texts of the translatable alarm properties are returned with input parameter `languages` in the operation request.

- `filterString: "path LIKE '*::motor1.currentSpeed:*'"`
  Supplies all active alarms of the `currentSpeed` element of the `motor1` structure tag.

See also

"activeAlarms" query (Page 8243)

"activeAlarms" subscription (Page 8245)

20.4.5.5 Other operations

"login" mutation

Requirements

- The login data of a UMC user configured for runtime is known.

Description of "login"

```
login (
     username: String
     password: String
   ): Session
```

Operation name	`login`
Operation type	mutation
Function	Logs the GraphQL client in to the GraphQL server with the login data of a UMC user configured for Runtime. The user remains logged in until the session is closed, the user is logged out or the connection is disconnected.

Input parameters	`username`
	• Mandatory parameter
	• Data type: String
	• The user name of a UMC user configured for Runtime
	`password`
	• Mandatory parameter
	• Data type: String
	• The password of the UMC user

Selection set	Mandatory specification Specify which attributes the server returns. You can request the attributes defined in the `Session` type. For information on the notation, see section Structure of a client query (Page 8235). Request at least `token`. You require `token` for subsequent server requests. You pass `token` in the HTTP header `Authorization`, with appended `Bearer` prefix.
Server response	Supplies the attributes of the `Session` instance requested in the selection set as key-value pairs of a JSON data record.
Error messages (`code: description`)	• `0: Success` • `101: Incorrect credentials provided` • `102: UMC error`

See also

Disconnection by server (Page 8266)

"extendSession" mutation (Page 8252)

Logging in to the GraphQL server (Page 8226)

"Session" type (Page 8260)

"loginSWAC" mutation

Requirements

- The login data of a UMC user configured for runtime is known.
- The GraphQL client obtained a valid `Nonce` instance by calling `nonce` from the server.
- The GraphQL client has logged in to UMC with its user name and password and transmitted the nonce to UMC when logging in. In response, UMC sent a claim with a built-in nonce as well as a signed claim with a built-in nonce with a private key.

Description of "loginSWAC"

```
loginSWAC(
      claim: String
      signedClaim: String
    ): Session
```

Operation name	`loginSWAC`
Operation type	mutation
Function	Transmits the claim and signed claim that the GraphQL client received when logging in to UMC to the GraphQL server. When the `Nonce` contained in `claim` or `signedClaim` is valid, the client is logged on to the server. The user remains logged in until the session is closed, the user is logged out or the connection is disconnected.

Input parameters	`claim`
	• Mandatory parameter
	• Data type: String
	• The claim submitted by UMC.
	`signedClaim`
	• Mandatory parameter
	• Data type: String
	• The SignedClaim submitted by UMC.

Selection set	Mandatory specification Specify which attributes the server returns. You can request the attributes defined in the `Session` type. For information on the notation, see section Structure of a client query (Page 8235). Request at least `token`. You require `token` for subsequent server requests. You pass `token` in the HTTP header `Authorization`, with appended `Bearer` prefix.

Server response	Supplies the attributes requested in the selection set of the `Session` instance generated for the login as key-value pairs of a JSON data record.
Error messages (`code: description`)	• `0: Success` • `101: Incorrect credentials provided` • `103: Nonce expired`

See also

"nonce" query (Page 8253)

"Nonce" type (Page 8258)

Disconnection by server (Page 8266)

"extendSession" mutation

Requirement

- The GraphQL client is logged in to the server with a valid UMC user.

Description

```
extendSession: Session
```

Operation name	`extendSession`
Operation type	mutation
Description	Extends the session of the GraphQL client. Call the operation to prevent clients that are mainly waiting for notifications from a subscription, for example, from becoming inactive and having to log in again.
Input parameters	-
Selection set	Mandatory specification Specify which attributes the server returns. You can request the attributes defined in the `Session` type. For information on the notation, see section Structure of a client query (Page 8235).
Server response	Supplies the attributes of the `Session` instance requested in the selection set as key-value pairs of a JSON data record.
Error messages (`code: description`)	-

See also

"login" mutation (Page 8250)

"Session" type (Page 8260)

"session" query

Requirement

- The GraphQL client is logged in to the server.

Description

```
session(
      allSessions: Boolean
    ): [Session]
```

Operation name	`session`
Operation type	query
Description	Queries the current session data from the server.
Input parameters	`allSessions` • Optional • Data type: Bool • Default: False: • True: The server sends the data for all sessions of the logged-in user.
Selection set	Mandatory specification Specify which attributes the server returns. You can request the attributes defined in the `Session` type. For information on the notation, see section Structure of a client query (Page 8235).
Server response	Supplies the attributes of the `Session` instance or instances requested in the selection set as key-value pairs of a JSON data record.
Error messages (`code: description`)	-

See also

"Session" type (Page 8260)

"nonce" query

Description

```
nonce: Nonce
```

Operation name	`nonce`
Operation type	`query`
Function	Queries a `Nonce` instance from the GraphQL server. The nonce is required for the SWAC login.
Input parameters	-

Selection set	Mandatory specification Specify which attributes the server returns. You can request the attributes defined in the `Nonce` type. For information on the notation, see section Structure of a client query (Page 8235). Request `value`.
Server response	Supplies the attributes of the `Nonce` instance requested in the selection set as key-value pairs of a JSON data record.
Error messages (`code: description`)	• `0: Success` • `101: Incorrect credentials provided` • `103: Nonce expired`

See also

"Nonce" type (Page 8258)

"loginSWAC" mutation (Page 8251)

20.4.5.6 Reference for Unified-specific types and enumerations

Introduction

This section describes the Unified data types used in the GraphQL client queries and server responses.

For information about the enumerations used in these types, see the interface documentation that is available online when schema introspection is enabled.

"ActiveAlarm" and "ActiveAlarmNotification" types

Description

Instances of the type `ActiveAlarm` represent the information about alarms that you can request in `activeAlarms` queries from the server. Instances of type `ActiveAlarmNotification` represent information that you can request through `activeAlarm` subscriptions.

This includes:

- Properties of active alarms
  The table below provides a reference for the alarm properties.
  Translatable properties are returned as a string array. The array contains the alarm property texts in the languages requested and the order defined with input parameter `languages` in the operation request.
- The `languages` attribute
  A string array with the languages that the user has specified with input parameter `languages` in the `activeAlarms` request.
  For correct processing of the strings returned in the arrays for translatable alarm properties.

Reference of the alarm properties

The following table lists the alarm properties provided in the `ActiveAlarm` and `ActiveAlarmNotification` types in alphabetical order:

Name	Data type	Description
`acknowledgmentTime`	`Timestamp`	`Timestamp` instance for the time at which the alarm was acknowledged
`alarmClassID`	`Int`	User-defined ID of the alarm class of the alarm
`alarmClassName`	`String`	Name of the alarm class of the alarm
`alarmClassSymbol`	`[String]`	The symbols configured for the alarm class Translatable property
`alarmGroupID`	`Int`	User-defined ID of the alarm group to which alarm belongs
`alarmParameterValues`	`[Variant]`	To dynamize the following texts: `alarmText1` to `alarmText9` these values can be used as parameters in these texts.

`alarmText1`
`bis`
`alarmText9`
	`[String]`	Additional texts for the alarm
		Translatable properties

`alarmType`	`[String]`	Supplies details regarding the alarm condition. The values can be user-defined or product-specific. Translatable property
`area`	`String`	Name of the area to which the alarm belongs
`backColor`	`Color`	Background color of the alarm in the alarm control
`changeReason`	`[AlarmChangeReason]`	Reason for changes to the alarm as defined in enumeration `AlarmChangeReason`
`clearTime`	`Timestamp`	`Timestamp` instance for the time at which the alarm became inactive
`connectionName`	`String`	Only for PLC alarms Name of the HMI connection
`deadBand`	`Variant`	Hysteresis for `valueLimit` for suppressing the noise component during alarm generation
`duration`	`Timespan`	The time interval between the time when the alarm became active and the time when the alarm state changed the last time
`eventText`	`[String]`	Main text for the alarm, typically describes the cause of the alarm Translatable property
`flashing`	`Boolean`	True: The alarm flashes in the alarm control
`hostName`	`String`	Name of the device that hosts the source of the alarm In the case of alarms triggered by operator input, this is the client on which the operator made the input.
`infoText`	`[String]`	More information for the operator, such as instructions or standard procedures for handling this alarm Translatable property
`instanceID`	`Int`	Alarm ID of the active alarm For unambiguous alarm identification of configured alarms that can have more than one active alarm instance at a time.
`invalidFlags`	`AlarmInvalidFlags`	`AlarmInvalidFlags` instance that indicates whether attributes or configuration of the alarm are invalid.

Name	Data type	Description
`loopInAlarm`	`String`	A script function that jumps to the screen that has triggered the alarm.
`loopInAlarmParameterValues`	`Variant`	Parameters for loop-in-alarms
`modificationTime`	`Timestamp`	`Timestamp` instance for the last time one of the properties of the alarm was changed
`name`	`String`	Name of the configured alarm
`notificationReason`	`String`	Only for `ActiveAlarmNotification` Reason for the notification Possible values: • "Added" An active alarm was added to the subscription. • "Modified" A subscribed alarm was modified. • "Removed" A subscribed alarm was removed from the subscription because it became inactive or no longer satisfies the filter criteria.
`origin`	`String`	Name of the object that activated this alarm instance, e.g. an HMI device or function block.
`path`	`String`	The full alarm name Can be used for filtering the `activeAlarm` query or subscription. See section Filtering alarms (Page 8248).
`priority`	`Int`	The current priority of the active alarm Possible values: • 0: No priority assigned • 1-5: Low (primarily an informative function) • 6-10: Medium (warnings) • 11-15: High (errors) • 16: Highest (critical alarms)
`producer`	`AlarmProducer`	The Siemens product or the function of a domain to which the alarm source belongs, as defined in enumeration `AlarmProducer`
`quality`	`Quality`	Quality of `state` as `Quality` instance
`raiseTime`	`Timestamp`	`Timestamp` instance for the time at which the alarm became active
`resetTime`	`Timestamp`	`Timestamp` instance for the time at which the alarm was reset
`sourceID`	`String`	Identifies the source of the alarm instance For controller alarms, the communication driver assigns the ID based on the reference of the external alarm source, such as the alarm area ID of the S7 PLC. For application alarms, the application or the service assigns the ID so that the alarm has a relationship with a service-specific object, e.g. recipe name/ID by the service of the parameter set control.
`sourceType`	`AlarmSourceType`	Type of source from which the alarm was generated, e.g. tag-based, PLC-based or system-based, as defined in enumeration `AlarmSourceType`
`state`	`AlarmState`	The current state of the active alarm as defined in enumeration `AlarmState`

Name	Data type	Description
stateMachine	AlarmStateMachine	State machine of the active alarm The state machine defines possible states of the alarm and transitions between these states, as defined in enumeration AlarmStateMachine
stateText	[String]	The user-defined description of the alarm state Translatable property
suppressionState	AlarmSuppressionState	Visibility of the alarm Shelved alarms are not visible in the alarm control but are still active, as defined in enumeration AlarmSuppressionState
systemSeverity	Int	Alarm severity from the perspective of the system The number of severe alarms in a system provides an indication of the health of the system. In redundant systems, the healthier host becomes the active host and the less healthy host becomes the passive host.
textColor	Color	Text color of the alarm in the alarm control
userName	String	Name of the user associated with the last alarm event
userResponse	AlarmUserResponse	The expected operator response as defined in enumeration AlarmUserResponse
value	Variant	The value of the alarm
valueLimit	Variant	Limit for value as of which the alarm becomes active. If a dynamic limit has been configured for the alarm, then the updated limit
valueQuality	Quality	Quality of value as Quality instance

See also

"activeAlarms" query (Page 8243)

"activeAlarms" subscription (Page 8245)

"AlarmInvalidFlags" type (Page 8257)

"Time span" type (Page 8261)

"Time stamp" type (Page 8262)

"Quality" type (Page 8259)

"AlarmInvalidFlags" type

Description

The AlarmInvalidFlags type defines attributes for invalid attributes or configurations of alarms.

Name	Data type	Description
invalidConfiguration	Boolean	The alarm has an invalid configuration.
invalidTimestamp	Boolean	The alarm has an invalid time stamp.

Name	Data type	Description
`invalidAlarmParameter`	`Boolean`	The alarm has an invalid alarm parameter.
`invalidEventText`	`Boolean`	The alarm has an invalid event text.

See also

"ActiveAlarm" and "ActiveAlarmNotification" types (Page 8254)

"Error" type

Description

The `Error` type defines the attributes of item-level errors.

For GraphQL operations with item-level errors, you can request these attributes from the server via the selection set when the operations call is made.

Attributes:

Name	Data type	Description
`code`	String	Numeric code of the error message
`description`	String	Detailed description of the error message

See also

Top-level and item-level errors (Page 8267)

"TagValueResult" type (Page 8261)

"WriteTagValueResult" type (Page 8263)

"TagValueNotification" type (Page 8260)

"Session" type (Page 8260)

"Nonce" type

Description

The `Nonce` type defines attributes of a nonce that are required to log in to a GraphQL client via SWAC login.

Attributes:

Name	Data type	Description
`value`	`String`	The nonce string
`validFor`	`Int`	Time span in seconds until the nonce instance expires

See also

"nonce" query (Page 8253)

"loginSWAC" mutation (Page 8251)

"Quality" type

Description

An instance of type `Quality` represents the quality of a tag value or alarm value (`value` attribute).

Attributes:

Name	Data type	Description
`quality`	`MainQuality`	The main QualityCode, as defined in enumeration `MainQuality`
`subStatus`	`QualitySubStatus`	The reason for the main QualityCode, as defined in enumeration `QualitySubStatus`
`limit`	`QualityLimit`	Information on whether or how the QualityCode is related to defined limit values, as defined in enumeration `QualityLimit`
`extendedSubStatus`	`QualityExtendedSubStatus`	More detailed information on `subStatus`, as defined in enumeration `QualityExtendedSubStatus`
`sourceQuality`	`Boolean`	The source that set `value` also set quality.
`sourceTime`	`Boolean`	The source that set `value` also set a time stamp.
`timeCorrected`	`Boolean`	The time stamp set by the source has been corrected.

"QualityInput" type

Description

The `QualityInput` type describes the attributes that are passed when `writeTags` is called in the input parameter `quality`.

Name	Data type	Description
`quality`	`MainQuality`	Mandatory specification The main QualityCode, as defined in enumeration `MainQuality`
`subStatus`	`QualitySubStatus`	The reason for the main QualityCode, as defined in enumeration `QualitySubStatus`

See also

"writeTags" mutation (Page 8239)

"Session" type

Description

The `Session` type defines attributes of a session.

Attributes:

Name	Data type	Description
`token`	`String`	ID of the authorization token for this session In subsequent server requests, pass the ID of the authorization token in the HTTP header `Authorization`, preceded by the `Bearer` prefix. When operations are called that require a user to be logged on to the server, this ID must be passed in the HTTP header `Authorization`, preceded by a `Bearer` prefix.
`expires`	`Timestamp`	Time at which the session expires, as `Timestamp` instance
`error`	`Error`	`Error` instance of possible item-level errors

See also

"session" query (Page 8253)

"UserGroup" type (Page 8262)

"User" type (Page 8262)

"login" mutation (Page 8250)

"extendSession" mutation (Page 8252)

"Error" type (Page 8258)

"TagValueNotification" type

Description

The `TagValueNotification` type defines which attributes the notifications for subscribed tags can contain.

Attributes:

Name	Data type	Description
`name`	`String`	Name of the subscribed tag
`value`	`Value`	`Value` instance of the tag

Name	Data type	Description
error	Error	Error instance of possible item-level errors
notificationReason	String	Possible values: • "Added" The subscription of the tag was started. The property values of the tag are sent at the start time of the subscription. • "Modified" The tag was modified. Your current property values are sent. • "Removed" The subscription of the tag was closed.

See also

"Error" type (Page 8258)

"tagValues" subscription (Page 8242)

"TagValueResult" type

Description

The TagValueResult type defines which attributes the server can supply in its response after the tagValues query is called.

Attributes:

Name	Data type	Description
name	String	Tag name
value	Value	Value instance of the tag
error	Error	Error instance of possible item-level errors

See also

"tagValues" query (Page 8238)

"Error" type (Page 8258)

"Value" type (Page 8263)

"Time span" type

Description

Type used to define a time span.

Data type of time span: RAW (integer in 100 ns)

See also

"ActiveAlarm" and "ActiveAlarmNotification" types (Page 8254)

"Time stamp" type

Description

Type for the definition of a time stamp according to ISO 8601 (https://en.wikipedia.org/wiki/ISO_8601).

See also

"ActiveAlarm" and "ActiveAlarmNotification" types (Page 8254)

"User" type

Description

The `User` type defines attributes of the user logged in to GraphQL.

Attributes:

Name	Data type	Description
`id`	`String`	User ID
`name`	`String`	User name
`groups`	`[UserGroup]`	The user group to which the users belong, as `UserGroup` instance
`fullName`	`String`	The full user name, such as first name and last name
`language`	`String`	The language set for the user in UMC Can be empty.
`autoLoggOffSec`	`Int`	Time period in seconds for which the user can be inactive before their authorization token expires

See also

"Session" type (Page 8260)

"UserGroup" type (Page 8262)

"UserGroup" type

Description

The `UserGroup` type defines attributes of a user group.

Attributes:

Name	Data type	Description
`id`	`String`	ID of the user group
name	`String`	Name of the user group

See also

"User" type (Page 8262)

"Value" type

Description

An instance of type `Value` represents a tag value.

Attributes:

Name	Data type	Description
`value`	`Variant`	The current process value
`timestamp`	`Timestamp`	Timestamp of the last change of `value`
`quality`	`Quality`	Quality of `value`

See also

"TagValueResult" type (Page 8261)

"Quality" type (Page 8259)

"TagValueNotification" type (Page 8260)

"WriteTagValueResult" type

Description

An instance of type `WriteTagValueResult` represents a feedback for the write access to a tag.

Attributes:

Name	Data type	Description
`name`	`String`	Tag name
`error`	`Error`	Error instance of possible item-level errors

See also

"writeTags" mutation (Page 8239)

"Error" type (Page 8258)

20.4.6 Code examples

Numerous code examples in multiple programming languages are included in the scope of delivery of WinCC Unified GraphQL.

Overview

Programming language	Covered topics
Python	Login Querying and changing data Subscribing to data
HTML and Javascript	Subscribing to data and displaying in Google diagrams
C#	Purely WebSocket-based subscription without GraphQL library
SWAC	Login with a SWAC-based login method

Installation

The GraphQL examples are an integral component of the Openness SDK and available on the WinCC Unified installation medium.

You can find information on installing the SDK in the "Installation SDK" user help.

20.4.7 Recommended procedures

20.4.7.1 Performance optimization

The performance capability of a GraphQL client depends on how the GraphQL API is being used.

Below are tips on how you can optimize the performance of your GraphQL application.

Limiting the selection

In its response to the client, the GraphQL server transmits the fields in the selection set that were specified by the client for the operation request. In most cases, fields that were not requested are not processed by the server.

Improve the performance of your client by using selection sets that are precisely tailored to your requirements.

Example

- `activeAlarms` query with extensive selection set:
```
query {activeAlarms(languages: ["en-US", "de-DE"]filterLanguage:
"en-US") {name instanceID alarmGroupID raiseTime
acknowledgmentTime clearTime resetTime modificationTime state
textColor backColor flashing alarmClassName alarmClassSymbol
alarmClassID stateMachine priority alarmParameterValues languages
alarmType eventText infoText alarmText1 alarmText2 alarmText3
alarmText4 alarmText5 alarmText6 alarmText7 alarmText8 alarmText9
stateText origin area changeReason connection valueLimit
sourceType suppressionState hostName userName value valueQuality
{ quality subStatus } quality { quality subStatus } invalidFlags
{ invalidAlarmParameter invalidConfiguration invalidEventText
invalidTimestamp } deadBand producer duration sourceID
systemSeverity loopInAlarm loopInAlarmParameterValues tag
userResponse } }
```
- `activeAlarms` query with limited selection set:
```
query { activeAlarms( languages: ["en-US", "de-DE"]
filterLanguage: "en-US" ) { name instanceID eventText } }
```

Bulk operations

Method calls of a web client have a larger overhead than method calls of locally connected DLLs or RPCs (Remote Procedure Calls) because a connection between the client and server has to be established in the background. This also applies to GraphQL.

Reduce this overhead by calling a method once and passing a list of objects as input parameter instead of individually calling the method for each object. The GraphQL server executes the operation for all objects from the list and supplies the result in a response.

Example

Objective	Read out the timestamp of multiple tags
Query	`query{ tagValues(names: ["Tag_1"],["Tag_2"],["Tag_3"]) {value {value timestamp} error { description }}`
Response	`{"data": {"tagValues": [{"value": {"value": 75, "timestamp": "2022-07-20T07:50:46.476" }, "error": {"description": "Succeeded" }}, {"value": {"value": false, "timestamp": "1970-01-01T00:00:00.000Z" }, "error": {"description": "Succeeded" }},{{"value": "value": false, "timestamp": "1970-01-01T00:00:00.000Z" }, "error": {"description": "Succeeded" }}]}}`

Tip for working efficiently

Change the same tag in the same mutation request multiple times.

Example: A GraphQL client is to document the values of a tag for audit purposes. The client subscribes to the tag and caches the reported value changes (value and timestamp). At regular intervals, the client executes a mutation to which it passes the cached values with their associated timestamps.

Using efficient client libraries

Libraries for GraphQL clients hide the details on HTTP and WebSockets from the developers. They convert the responses of the server into objects that correspond to the programming language, check whether the client code violates the schema, and more. This simplifies programming of a GraphQL client but slows down its execution.

How useful and performant a library is depends on which functions the library takes on, how well it is programmed and the programming language used.

Test a variety of libraries in advance before deciding which library is best suited to your needs.

Using a custom client

If performance is the most important factor, you can use the GraphQL API directly with an HTTP and/or WebSocket library instead of a client library.

Queries and mutations are HTTP POST messages in which the operation is sent in the POST body and the authorization in the HTTP header.

Subscriptions are WebSocket connections. Their input parameters are transferred as WebSocket messages.

You can examine the details of the messages, e.g. with the Google Chrome developer tools, while operations are executed in Apollo Studio.

See also

Filtering alarms (Page 8248)

20.4.7.2 Disconnection by server

Overview

Normally, the client closes the connection to the GraphQL server by requesting the `logout` mutation. In rare cases, the server closes the connection, e.g. when it is stopped.

When the server closes the connection, the GraphQL client library receives a notification regarding this. The client must reestablish the connection. If the server is still running, the client must execute the subscription again.

Handling of subscriptions after connection reestablishment

If the server is still running, the client can execute the subscription again.

If the server has been stopped, the client must cyclically query the server until it is available again, e.g. by executing the subscription operation until it is successful.

Examples

- Folder "Support\Openness\GraphQL\Examples\PythonClient" on the WinCC Unified installation medium contains the following code example: "gql_subscription.py"
  In the example, the iterator is closed and a specific exception is thrown.
- The following Python code snippet shows how the client closes the subscription and how it is notified when the server closes the subscription:

```
try:

  # Using `async with` on the client will start a connection on the transport
  # and provide a `session` variable to execute queries on this connection
  async with Client(
      transport=transport,
      fetch_schema_from_transport=True,
  ) as session:
      async for result in session.subscribe(subscription):
          print(result)

          counter += 1
          # this is just a dummy condition for exiting the loop and therefore closing the subscription
          # a real life condition would be e.g., checking a bool set from outside, when the user
          # clicks an "x" button to close the subscription window
          if counter == 10:
            print ("Closing connection from client side")
            break

except ConnectionClosed:
  print ("Connection closed by the server")
```

See also

"login" mutation (Page 8250)

20.4.8 Troubleshooting

20.4.8.1 Top-level and item-level errors

Error groups

The following error groups exist in GraphQL:

- Top-level errors
  The entire method request has failed. Only the error message is supplied, and no result data.
- Item-level errors
  The method was able to be requested, but parts of the method call were not successful. Result data is supplied. The parts of the method request that were not successful contain an error code other than "0" in the `error` attribute and an error description other than "Succeeded".

Note

Error attributes specified in the selection set

In order for the server response to contain error information about item-level errors, you must request the desired error attributes when calling the operation with the selection set.

It is recommended that the `code` or `description` attribute, or both, be requested.

Example for a top-level error

If the requested language is invalid, this results in the following error message:

```
{
  "errors": [
  {
      "message": "At least one of the requested languages is
invalid.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }],
      "path": ["activeAlarms"],
      "extensions":
      {
        "code": "INTERNAL_SERVER_ERROR"
      }
  }],
  "data":
  {
    "activeAlarms": null
  }
}
```

Attributes:

- `message`: Easy-to-understand error description
- `locations` and `path` The part of the query that contains errors
- `code`: A general or specific error code
  `INTERNAL_SERVER_ERROR` is the default error code if no specific error code exists.
- (when development mode is enabled) `stackTrace`: For further error diagnostics

Example for an item-level error

A writeTagValues call is to write values to two tags. The write operation was able to be successfully performed for tag "TrainingString1" but not for tag "Intern_Int_2". The result data for "Intern_Int_2" contains the error code and an error description. The reason for the error is that the passed data type (String) could not be converted to the expected data type (DTInt):

```
{
  "data": {
    "writeTagValues": [
      {
        "name": "TrainingString1",
        "error":
        {
          "code": "0",
          "description": "Succeeded"
        }
      },
      {
         "name": "Intern_Int_2",
         "error":
         {
           "code": "201",
           "description": "Can't convert provided value to data type
'DTInt': 'failing string value'"
         }
       }]
  }
}
```

See also

Structure of a client query (Page 8235)

"Error" type (Page 8258)

20.4.8.2 GraphQL server doesn't start

Problem	GraphQL server doesn't start
Identify the cause	1. In the installation folder of WinCC Unified, double-click the "RTILtraceViewer.exe" file under "WinCCUnified\bin". The "TraceViewer" application is started. 2. Browse the traces for the trace entry documenting the failed start of the GraphQL server.
Examples of possible causes	• The port configured for the GraphQL server is being used by another application. • A configured plug-in could not be loaded.

Port occupied

If the port needed for the GraphQL server is being used by another application, the GraphQL server cannot start.

Resulting trace entry:

- Severity "Fatal"
- Message: "Port 4000 already in use. Exiting..."

Remedy: Select a different port for the other application.

Index

U

V