SIEMENS

COMOS

Platform
Class documentation ivbQuery_dll

Programming Manual

09/2014
A5E32035467-AA

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
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E32035467-AA
℗ 09/2014 Subject to change

Table of contents

1 Interfaces for Query

1.1 Basic functions

1.1.1 Introduction

The following subs and functions are all programmed the same way, but do not derive from a common base object:

- Refresh (Page 9)
- ShutDown (Page 10)
- Copy (Page 10)
- Version (Page 10)
- IsChanged (Page 10)
- Dispatch (Page 10)
- Storage (Page 11)
- Owner (Page 11)
- Locked (Page 11)
- Collection (Page 12)

1.1.2 Refresh

Refresh

```
Refresh()
```

Refreshes the relevant component.

Note

In general, Refresh is not the same as a recalculation. During Refresh the data is read and displayed afresh. During a recalculation, the OrigCollection is recalculated, which generally does not happen.

If Refresh also contains a recalculation, this is explicitly mentioned, e.g. at RefreshRow (Page 72).

1.1.3 ShutDown

ShutDown

```
ShutDown()
```

All interfaces have a ShutDown.

Note

A ShutDown recursively calls all items of the Childs collection and "eliminates" them. You can no longer work with the "eliminated" objects.

1.1.4 Copy

Copy

```
Copy() as Object
```

Generates a copy of the current instance.

1.1.5 Version

Version

```
Version() as Integer Read only
```

Current version of the interface.

1.1.6 IsChanged

IsChanged

```
IsChanged() as Boolean
```

Checks whether the instance was changed (e.g. because a property was implemented).

1.1.7 Dispatch

Dispatch

```
Dispatch() as Object Read only
```

Switches from the interface to the relevant implementation. With that, even the properties that are not part of the interface itself are available.

Is required especially in scripts because the relevant implementations are available only then.

Example:

ITopQuery is implemented in ten different components (Implementations of ITopQuery (Page 17)), among other things, also in TopQDevices. If this implementation is active, the TopQDevices properties become available through Dispatch. Then the following would be possible: TopQuery.Dispatch. Class, whereby Class comes from TopQDevices. Naturally this is only possible if the correct implementation is active.

1.1.8 Storage

Storage

```
Storage() as IStringStorage Read only
```

Switches to the IStringStorage (Page 59) interface.

Storage has a certain similarity with `Dispatch`, because `Storage`, too, provides methods and properties that do not come from the current interface.

All components that have `ITopQuery` implemented also implement `IStringStorage`.

1.1.9 Owner

Owner

```
Owner() as Object Read only
```

Returns the owner. If a specific object type is required as owner, the declaration is limited accordingly.

1.1.10 Locked

Locked

```
Locked() as Boolean
```

The instance must not be changed, either in the user interface or in the script or in any other way.

Example

An instance of `IQuery` is a query object.

1.1.11 Collection

Collection

In the queries, independent collections are used that have nothing to do with the collection objects of the COMOS kernel.

- As a rule, the collections of queries are `Read only` and know only `Item` and `Count`.
- If an `Add` is possible, it is stated at the appropriate places.

1.2 Important spellings / parameter

Default value "-1"

The default value "-1" can appear only for optional parameters and detected, depending on the context, all elements / the last element / the end of collection. What exactly applies is specified at the relevant location.

If an index is mandatory, then a concrete index must be specified and you cannot work with "-1".

Index: Start value

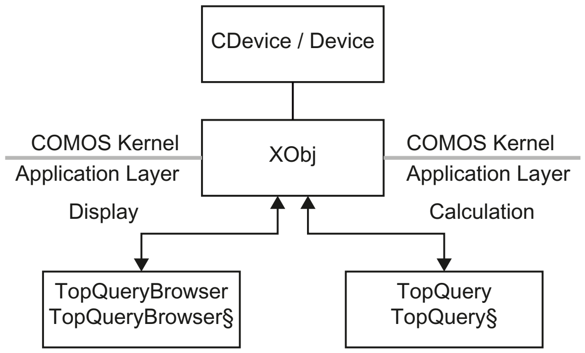
All listings or collections in Query that have an "index" must start with 1.

1.3 Graphics

1.3.1 Figure 1: Overview of the class hierarchy

Introduction

This following figure describes how the object queries are integrated in the Comos object hierarchy.



Example

All object queries use a Device or CDevice as container. Hence, in the script, an object query can be called as follows:

```
Device.Xobj.TopQuery
```

The query is loaded only now and the properties etc. can only be used from now on.

Entry possible at any level

It is not absolutely necessary to start at the container – the Device or CDevice. It is also possible, for example, to work directly with TopQuery or TopQueryBrowser. In this case, the developer must worry about a functional hierarchy.

Separation of display and calculation

In the Query hierarchy, the display and calculation are separated:

- TopQuery
  The calculation component can exist by itself, without display in the Comos user interface. From the container, you receive the component through: `Device.Xobj.TopQuery.`
- TopQueryBrowser
  This component cannot work practically by itself, rather must always work together with TopQuery. It is compulsory to use TopQueryBrowser if you want to display and open a query in Comos. However, TopQueryBrowser does not need to be opened from the container.
  Example:
  The project selection also uses TopQueryBrowser, but without a device container (at this point, no device is loaded yet). From the container, you receive the component through: `Device.Xobj.TopQueryBrowser`

Using queries

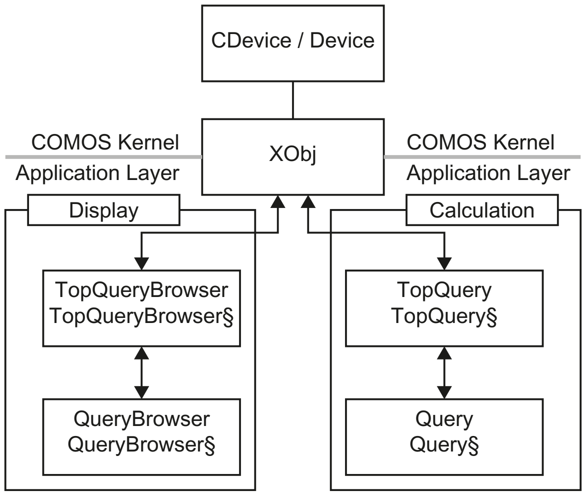
If a query is used in a report, the report knows the query as a ReportObject.

It does not work vice-versa. A query never knows its uses – and consequently, a query also does not know the report where it is used. Another example is processing a query in a script; here, too, the query does not know the script that calls the query.

1.3.2 Figure 2: Single structure of the class hierarchy

Structure

Both the display side on the left and the calculation side on the right are each divided further:



- TopQueryBrowser calls TopQuery, thus:
  `Device.Xobj.TopQuery.Query`
- TopQuery calls Query, thus:
  `Device.Xobj.TopQueryBrowser.QueryBrowser`

The top level provides additional functions for the ten implementations (Implementations of ITopQuery (Page 17)). The levels underneath (QueryBrowser and Query) provide the functionalities that are always needed.

The same applies here: It is possible to enter at any level. It is allowed to generate a `Query` without a `TopQuery`.

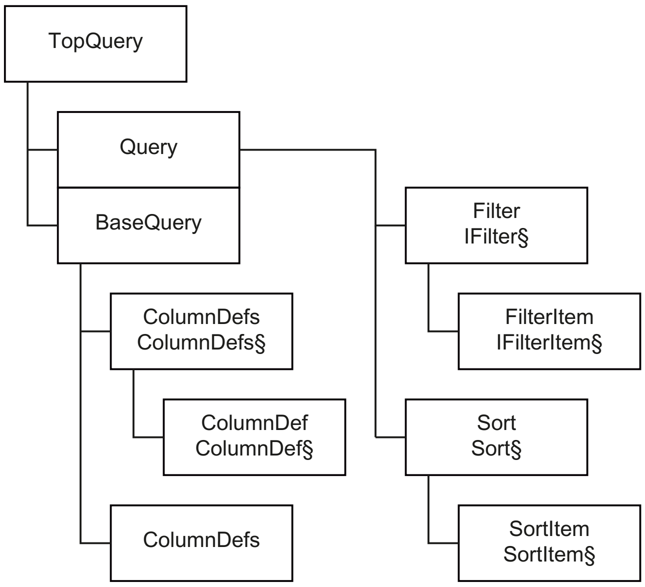
Example of project selection

At the time of project selection, no Devices or CDevices are loaded. Thus, there is no possibility to load a query through the Device container. The top level functions are also of no use. So, the project query creates a QueryBrowser, generates a Query for it and assigns it to the QueryBrowser.

1.3.3 Figure 3: Owner hierarchy

The interface for Query has a separate owner hierarchy. At different points in this documentation, you will find a property with the name "Owner". An example there is the following property: IFilter.Owner. This property belongs to IFilter and the owner of IFilter is IQuery.

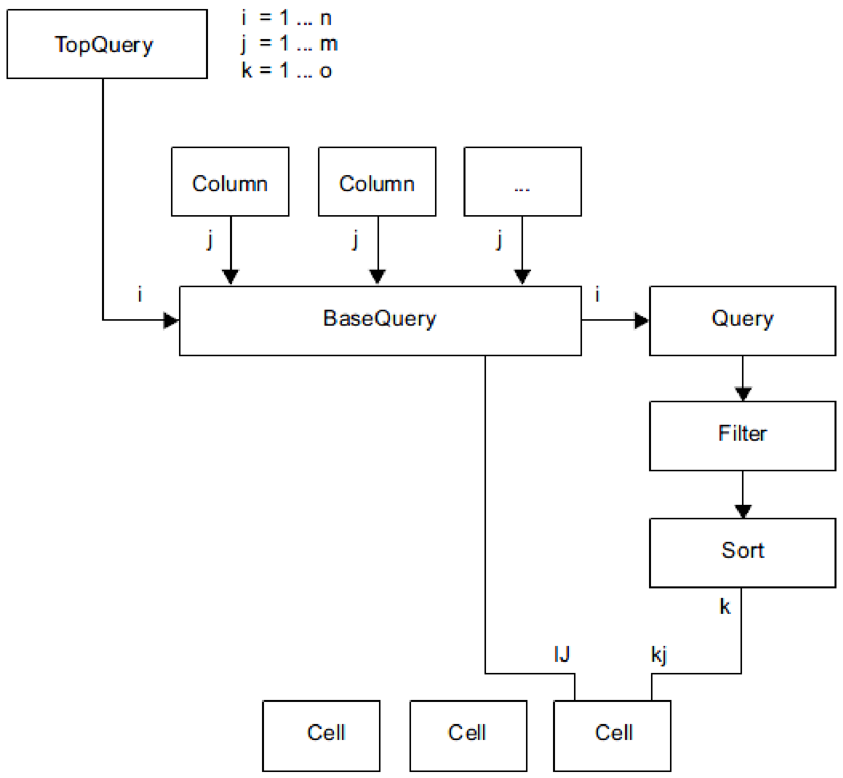
This information is can also be taken from the figure below:



The Filter is underneath Query, exactly as described in the definition of IFilter. Owner (Page 32).

1.3.4 Figure 4: Cell calculation without hierarchy

If you want to access a cell, you need the cell index. This index depends on from where the cell is calculated:



In the above figure, you can see different columns that are identified by means of the counter "j". The rows come from the OrigCollection and are likewise identified by means of a counter.

One can see:

- As long as no hierarchy is used, the column index is retained. If nested query hierarchies were introduced in the above figure, you would need to introduce another, variable column index.
- The row index first comes from the OrigCollection and is also available for BaseQuery. If you then filter and sort the query, a new row index is created.
- Regardless of whether you are calculating the cell from BaseQuery or from Query: You will get the same cell object.
- In the query environment, the indices always begin with 1.

1.4 ITopQuery

Task

Main task of ITopQuery is to contribute to the calculation of the OrigCollection. This is the basic set of objects with which the query works in the following.

OrigCollection

The OrigCollection is the result of the relevant scan procedure of the implementations (*Implementations of ITopQuery*). At this point, no filter or sorting etc. are active yet. The number of objects found corresponds to the number of rows in the QueryBrowser before the results are filtered, hidden etc. The scan procedures are stipulated for Query.

Example `TopQDevices`:
The Scan procedure are provided by the ScanManager.

The OrigCollection only knows the start object MainObject (Page 19).

Implementations of ITopQuery

At present, the interface `ITopQuery` is implemented at 10 components. Each of these implementations can have its own parameters, which are included in the actual calculation of the OrigCollection. So, the OrigCollection is not calculated in ITopQuery; the calculation varies depending on the implementation.

`TopQDevices, TopQCDevices, TopQDocuments, TopQSpecifications, TopQConnectors, TopQStdTables, TopQStdValues, TopQTranslate, TopQReimport, TopQGeneral.`

All instances that implement the ITopQuery, also implement `IStringStorage`.

1.4.1 ITopQuery: Sub

Init

```
Init()
```

Checks and initializes all set properties. This also includes the properties that come from the implementations. Then generates the Query object (this is the object that is seen in Figure 2: Single structure of the class hierarchy (Page 14)). The error handlers are part of the implementation.

Init independently tries to run the initialization without errors. Example: Query for engineering objects (TopQDevices), scan procedure: "BackPointer procedure". This procedure requires as set base object ("CObject"). Error correction through Init: If no CObject is set, the direct procedure is started ("Not recursive") instead of the BackPointer procedure.

Remark: If the query is started via the browser, it is impossible that no CObject exists, since the browser does not start the search without CObject.

See also *Execute*.

Refresh

`Refresh()`

Refresh (Page 9). Here Refresh from: TopQuery and Query.

Execute

`Execute()`

This sub combines *Init* and GetOrigCollection (Page 18) and fully calculates the query (i.e. including filter, etc.). Main purpose of this function is to use the query without QueryBrowser, but at the same time to calculate the query as if it were started in QueryBrowser.

ClearRows

`ClearRows()`

Empties the OrigCollection (Page 17) (OrigCollection = Nothing).

ShutDown

`ShutDown()`

ShutDown (Page 10). The children of this interface are: Query objects.

1.4.2 ITopQuery: Function

GetOrigCollection

`GetOrigCollection() as Object`

GetOrigCollection returns the OrigCollection (Page 17), so that both cannot differ from each other.

See also Execute (Page 17).

Copy

`Copy() as ITopQuery`

Copy (Page 10). Here, copy of: TopQuery.

Reasonable

`Reasonable(ByVal ItemObject as Object) as Boolean`

- `ItemObject`: To be tested object.

Checks whether an object matches the OrigCollection (Page 17) (but not if it is part of subset).

Example: The OrigCollection of TopQDevices may contain only Devices. Reasonable would deliver True, if an object is a Device. True would also be delivered if the Device would not be part of the OrigCollection.

1.4.3 ITopQuery: Property

cDescription

```
Description() As String
```

Description for a standard query.

Version

```
Version() as Integer Read only
```

Version (Page 10).

Query

```
Query() as IQuery
```

This is the Query object. See figure 2: Single structure of the class hierarchy

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10). Here, check for change of: TopQuery.

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10). Here: ITopQuery on the respectively currently used implementation (see sectionITopQuery (Page 17)).

Partners

```
Partners() as IPartnersRead only
```

For internal purposes.

Basically, TopQuery is responsible for serializing other instances, i.e. "to revive" them. For this, TopQuery has to know all required properties, constants, variabes, etc. in order to properly restore the object.

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as ObjectRead only
```

Owner (Page 11). Here: XObj. See Figure 2: Single structure of the class hierarchy (Page 14).

MainObject

`MainObject() as ObjectGet, Let, Set`

Can be a single IComosBaseObject, an IComosDCollection or even a VBA collection. Corresponds to the "Start object" field in the user interface (TopQueryBrowser).

NewItemObject

`NewItemObject() as ObjectRead only`

For internal purposes.

Is used by the query when new objects are generated.

1.5 IQuery

An instance that implements IQuery also implements IBaseQuery and IStringStorage. At present, this is only one implementation: Query.

1.5.1 IQuery: Sub

Refresh

`Refresh()`

Refresh (Page 9). Here Refresh from: Query.

ActionExecute

`ActionExecute(Optional ByVal QueryBrowser as IQueryBrowser)`

- `QueryBrowser:` the object query in Comos for which the script is to be executed.

If the implementation is executed together with the user interface, e.g. if the query is started as container through a Device, then QueryBrowser is available, too. However, Query does not know the QueryBrowser, see Figure 2: Single structure of the class hierarchy (Page 14). Therefore, the QueryBrowser must be set here explicitly.

At the Query object there is a script. ActionExecute executes this script.

Corresponds in the interface to: The "Execute" command in the top icon bar of the query or the exclamation mark in the script window on the bottom.

In addition, Query can be started without QueryBrowser (see explanation above). In this case, QueryBrowser must not be passed as parameter. For Query, it makes no difference if there is no MPE. But in the script, such an illegal procedure would produce errors.

See also ActionText (Page 23).

RefreshRow

`RefreshRow(ByVal Index as Long)`

- `Index`: Row index (ranging from 1 to IQuery. RowCount (Page 23)).

Updates the content of the row.

RefreshCell

```
RefreshCell(ByVal RowIndex as Long, ByVal ColIdent as
Variant)RowIndex: From 1 to IQuery.RowCount.
```

- `ColIdent`: (Integer or String): Either column index (ranging from 1 to IQuery. ColumnCount (Page 23)) or column name. Input difference: see Name (Page 42).

Updates the content of the cell.

RefreshSum

```
RefreshSum(Optional ByVal ColIdent as Variant)
```

- `ColIdent`: (Integer or String): Either column index (ranging from 1 to IQuery. ColumnCount (Page 23)) or column name. Input difference: see Name (Page 42).

Updates the column sum. See also Sum (Page 22).

ShutDown

```
ShutDown()
```

ShutDown (Page 10). The children of this interface are: Query objects.

ExportData

```
ExportData(ByVal ExportType as qeExportType, Optional ByVal FileName
as String, Optional ByVal TableName as String)
```

- `ExportType:` qeExportType (Page 115)
- `FileName`: User input for the file name under which the data is to be exported.
- `TableName`: can additionally be specified for data types that further distinguish within their files.

Example: Access: Filename of mdb, TableName: Name of table in the DB.

The query is exported in the data format specified by ExportType.

EvalByValue

```
EvalByValue(Optional ByVal RowIndex as Long = -1)
```

- `RowIndex`: Row index (ranging from 1 to IQuery. RowCount (Page 23)); -1: All (= calculate complete query)
  See also Default value "-1". (Page 12)

This function can only be called, if the calculation of a new object in the query is based on user input, whereas the user input cannot be an object.

EvalByValue can either calculate the current row or the entire query. Example: If immediate calculation of new objects is activated in the user interface (QueryBrowser), then EvalByValue is called with the row index. If the calculation mode is set to "calculate later", EvalByValue is called for the entire query.

EvalByValue does not abort in case of an inconsistency, but, depending on the mode, also calculates the subsequent rows if a row cannot be calculated.

1.5.2 IQuery: Function

Cell

```
Cell(ByVal RowIndex as Long, ByVal ColIdent as Variant) as ICell Get
only
```

- `RowIndex`: From 1 to IQuery. RowCount (Page 23)
- `ColIdent`: (Integer or String): Either column index (ranging from 1 to IQuery. ColumnCount (Page 23)) or column name. Input difference: see *Name*.
- ICell: Section ICell (Page 54)
  Returns the cell object.

ColumnBaseIndex

```
ColumnBaseIndex(ByVal ColIndex as Integer) as Integer
```

- `ColIndex`: Column index (ranging from 1 to IQuery. ColumnCount (Page 23)). Corresponds to ColIdent, if an integer is passed there.

Conversion to the ColumnIndex of the *IBaseQuery*. Background: For "flat" queries there is no difference between the ColIndex of the IQuery and the ColumnIndex of the IBaseQuery, because each shifting of a column is also reproduced in IBaseQuery.

Exception: Hierarchies are generated only in IQuery, so that there will be differences.

RowBaseIndex

```
RowBaseIndex(ByVal RowIndex as Long) as Long
```

- `RowIndex`: From 1 to IQuery. RowCount (Page 23)

The RowIndex is converted into the RowIndex of IBaseQuery, i.e. into the row index with regard to a row in the OrigCollection. Differences are produced through sorting, filters and intervention of IExtended (Page 61).

Copy

```
Copy() as IQuery
```

Copy (Page 10). Here, copy of: Query

SubQuery

```
SubQuery(ByVal RowIndex as Long) as IQuery
```

- `RowIndex`: From 1 to IQuery. RowCount (Page 23)

Occurs only in queries that have a hierarchy. To produce a hierarchy, sub-queries are produced.

Sum

```
Sum(ByVal ColIdent as Variant) as Double
```

- ColIdent: (Integer or String): Either column index (ranging from 1 to IQuery. ColumnCount (Page 23)) or column name. Input difference: see Name (Page 42).

Returns the sum that was generated on RefreshSum (Page 20).

MoveRows

```
MoveRows(ByVal BeforeIndex as Long, ByVal ToMoveIndexs as Variant)
as Long
```

- `BeforeIndex`: "Old index"
- `ToMoveIndexs`: Collection of indices of the to be moved rows. The indices need do not be consecutive, i.e. it is permitted to have "gaps" between the selected rows. At the target, the rows are inserted consecutively, i.e. here the indices are recalculated consecutively.

Moves one or more row(s), and accordingly adjusts the the row index in IBaseQuery. See also Index: Start value. (Page 12)

1.5.3 IQuery: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Ignore

```
IgnoreHierarchie() As Boolean
```

Ignoring the evaluation or non evaluation of the hierarchy. The default value is "False".

BaseQuery

```
BaseQuery() as IBaseQuery
```

Interface is switched from IQuery to IBaseQuery, so that the functions, properties etc. of IBaseQuery are available. Example: IQuery.BaseQuery.OrgCollection

Filter

```
Filter() as IFilter Get, Let, Set
```

Returns the filter object. Counterpart is *Sort*.

Sort

```
Sort() as ISort Get, Let, Set
```

Returns the sort object. Counterpart is *Filter*.

RowCount

```
RowCount() as Long Read only
```

Returns the number of rows. Is used in ColumnIndex as Max counter.

ColumnCount

```
ColumnCount() as Integer Read only
```

Returns the number of rows. Is used in ColumnIndex as Max counter. The number of columns always refers to all columns and not only to the visible columns.

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

ActionText

```
ActionText() as String
```

Returns the script from the object query. ActionExecute (Page 20) executes ActionText.

Locked

```
Locked() as Boolean
```

Locked (Page 11).

StyleType

```
StyleType() as qeStyleType
```

See qeStyleType (Page 114).

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as Object Read only
```

Owner (Page 11).

Here: For main query, the owner is ITopQuery. In a subQuery, the "preceding" query is the owner.

PictureType

`PictureType() as qePictureType`

See qePictureType (Page 115).

OwnerRowIndex

`OwnerRowIndex() as Long Read only`

The row index of Owner (Page 11).

See also Index: Start value (Page 12).

AllowAddNew

`AllowAddNew() as Boolean`

Allows the user to create a new row within a query.

IsRowValid

`IsRowValid(ByVal Index as Long) as Boolean Read only`

- `Index`: Row index (ranging from 1 to IQuery.*RowCount*).

This property specifies whether a row has already been recalculated by EvalByValue (Page 20). Returns False if:

- Row has not been calculated yet or
- Row could not be calculated.

IsValid

`IsValid() as Boolean Read only`

This property specifies whether a row was already recalculated by EvalByValue (Page 20) fully. Delivers False, if:

- the table was not yet calculated fully or
- at least one row could not be calculated correctly.

RowObject

`RowObject(ByVal Index as Long) as Object Read only`

- `Index`: Row index (ranging from 1 to IQuery.*RowCount*).

Returns the object of the row. The object must be take the object from the OrigCollection. For this, the index of the row in the OrigCollection must be known. Thus, RowObject corresponds to:

```
Query.BaseQuery.OrigCollection.Item(Query.RowBaseIndex(RowIndex))
```

HCount

`HCount() as Integer Read only`

Returns the number of hierarchy levels. Prerequisite for hierarchies in queries are ISortItem. GroupLevel (Page 37).

GlobalDictionary

`GlobalDictionary() as IDictionary Read only`

- IDictionary: Typ MS.Scripting.Dictionary

Returns values or objects of the MS Dictionary. This way you can use global variables.

ScriptText

`ScriptText() as String`

Returns the script block of the "Script" tab in the options with the following subs:

- AfterLoad
- AfterGetOrigCollection
- BeforeShutdown

1.6 IBaseQuery

1.6.1 IBaseQuery: Sub

Refresh

`Refresh()`

Refresh (Page 9).

RefreshRow

`RefreshRow(ByVal Index as Long)`

- Index: Row index (ranging from 1 to IBaseQuery. RowCount (IBaseQuery) (Page 28)).

Updates the row.

RefreshCell

`RefreshCell(ByVal RowIndex as Long, ByVal ColIdent as Variant)`

- `RowIndex`: Row index (ranging from 1 to IBaseQuery. RowCount (IBaseQuery) (Page 28)).
- `ColIdent`: (Integer or String): Either column index (ranging from 1 to IBaseQuery. ColumnCount (Page 28)) or column name. Input difference: see Name.

Updates the cell.

SetExtendedObject

```
SetExtendedObject(ByVal ExtObjectProgID as String, ByVal ExtObject
as IExtended)
```

- `IExtended`: See section: IExtended (Page 61)
- `ExtObjectProgID`: Returns the ProgID of the Extended object
- `ExtObject`: Returns the Extended object

At this point, the Extended object must already exist. See also GetExtendedObject. See also ExtendedObject.

EvalByValue

```
EvalByValue(Optional ByVal RowIndex as Long = -1)
```

- `RowIndex`: Row index (ranging from 1 to IBaseQuery. RowCount (IBaseQuery) (Page 28));
  -1: All

See also Default value "-1" (Page 12).

Is called by IQuery. EvalByValue (Page 20).

1.6.2 IBaseQuery: Function

Cell

```
Cell(ByVal RowIndex as Long, ByVal ColIdent as Variant) as ICell
```

- `ICell`: Siehe Abschnitt ICell (Page 54)
- `RowIndex`: Row index (ranging from 1 to IBaseQuery. RowCount (IBaseQuery) (Page 28)).
- `ColIdent`: (Integer or String): Either column index (ranging from 1 to IBaseQuery. ColumnCount (Page 28)) or column name.
  Input difference: see Name.

Is called by IQuery. Cell (Page 22).

InsertRow

```
InsertRow(ByVal MainRowObject as Object, Optional ByVal Index as
Long = -1) as Long
```

- `MainRowObject`: The row object
- `Index`: either row index (ranging from 1 to IBaseQuery. RowCount (IBaseQuery) (Page 28)) or default value –1 for end of collection
  See also Default value "-1" (Page 12).

Creates a new row in the OrigCollection (Page 17) at the index position specified at Index. If Index –1, then the new row is appended.

DeleteRow

```
DeleteRow(ByVal Ident as Variant) as Boolean
```

- `Ident`: Either the index of the object in the OrigCollection or the object itself.

Removes a row from the query (in IQuery). Naturally, the row object is retained.
See also Index: Start value (Page 12).

Copy

```
Copy() as IBaseQuery
```

Copy (Page 10).

CreateNewRow

```
CreateNewRow() as Object
```

Creates a new row at the end of the query. Then, EvalByValue (Page 26) is called for the new row.

GetExtendedObject

```
GetExtendedObject() as IExtended
```

- `IExtended`: See section IExtended (Page 61)

At this point, the Extended object must already exist.
See also SetExtendedObject (Page 26).

1.6.3 IBaseQuery: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

OrigCollection

```
OrigCollection() as Object
```

Returned by ITopQuery for the standard queries (Implementations of ITopQuery (Page 17)).
In all other cases, the developer is responsible.

Can be a single IComosBaseObject, an IComosDCollection or even a VBA collection.

Columns

```
Columns() as IColumnDefs
```

Collection of the originally created columns. What the OrigCollection is for the rows, Columns is for the columns.

RowCount (IBaseQuery)

```
RowCount() as Long Read only
```

Like RowCount (Page 23) for IQuery, but here for the OrigCollection (Page 17).

ColumnCount

```
ColumnCount() as Integer Read only
```

Redundant information, corresponds to Columns.Count.

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

ExtendedObject

```
ExtendedObject() as String
```

Alternative to SetExtendedObject (Page 26). Here, the ProgID of a user-defined class is returned, which is then used to create an ExtendedObject independently.

Locked

```
Locked() as Boolean
```

Locked (Page 11).

IsRowValid

```
IsRowValid(ByVal Index as Long) as Boolean Read only
```

- `Index`: Row index of the row that is to be tested (ranging from of 1 to IBase-Query.*RowCount (IBaseQuery)*)

IQuery calls IBaseQuery with converted indices.

IsValid

```
IsValid() as Boolean Read only
```

EditType

```
EditType() as qeEditType Read only
```

Specifies, whether the EvalByValue (Page 20) mode is active or not.

MappingInfo

```
MappingInfo() as String
```

Can only be used in connection with XML connectors.

See section IMapping (Page 94).

RowObject

`RowObject(ByVal Index as Long) as Object Read only`

Index: Row index (ranging from 1 to IBaseQuery.*RowCount (IBaseQuery)*).

Returns the row object.

SystemExtendedObject

`SystemExtendedObject(ByVal ProgID as String) as IExtended`

- `ProgID`: ProgID of the Extended object.

Extended objects that are used for Comos-internal purposes.

Mapping

`Mapping() as IMapping Read only`

Is used for XML connectors. See section: IMapping (Page 94).

PermissionsCheck

`PermissionsCheck() as Boolean`

Inactive. Checks whether it is permissible to process a Comos object. Will be used in future Comos versions for the rights administration.

1.7 IFilter

An object that implements IFilter, also implements IStringStorage (Page 59). Current implementations: QFilter.

1.7.1 IFilter: Sub

ShutDown

`ShutDown()`

ShutDown (Page 10).

Clear

`Clear()`

Deletes all items, thus all Filter objects.

1.7.2 IFilter: Function

Count

```
Count() as Integer
```

Number of FilterItems. See IFilterItem (Page 33).

Item

```
Item(ByVal Index as Integer) as IFilterItem
```

- `Index`: Index of the FilterItem, starts with 1.

In the user interface, an item corresponds to a row in the "Sorting / Filter" dialog.

See IFilterItem (Page 33)

AddNew

```
AddNew(Optional ByVal Index as Integer = -1) as IFilterItem
```

- `Index`: Either the index for which the new FilterItem is to be used (adds 1 to the subsequent indices). If -1, it is appended at the end of collection.
  See also Default value "-1", Index: Start value. (Page 12)

Creates a new FilterItem. The FilterItem object is returned.

See IFilterItem (Page 33)

Delete

```
Delete(ByVal ItemIdent as Variant) as Boolean
```

- `ItemIdent`: Either Index of FilterItem or FilterItem object

Deletes a FilterItem.
See also Index: Start value (Page 12).

Eval

```
Eval(ByVal BaseQuery as IBaseQuery, ByVal RowIndex as Long) as
Boolean
```

- `BaseQuery`: See IBaseQuery (Page 26).
- `RowIndex`: From 1 to IBaseQuery. RowCount (IBaseQuery) (Page 28)

The Filter statement must be evaluated for each row in order to be able to decide whether a row is hidden by the filter. FilterItem here refers to a cell. Reason: The FilterItem itself is defined for a column and is then applied here for a row in the RowIndex. Row and column produces cell.

Copy

```
Copy() as IFilter
```

Copy (Page 10). Here, copy of: Filter object

Add

```
Add(ByVal ItemObject as IFilterItem, Optional ByVal Index as Integer
= -1) as Integer
```

- `ItemObject`: See IFilterItem (Page 33).
- `Index`: Either the index for which the temporarily saved FilterItem is to be used (adds 1 to the subsequent indices) or end of collection.
  See also Default value "-1", Index: Start value (Page 12).

Inserts a temporarily saved FilterItem, either at the specified position or by appending it at the end of the collection (if -1).

Remove

```
Remove(ByVal ItemIdent as Variant) as IFilterItem
```

- `ItemIdent`: Either Index of FilterItem or FilterItem object
- `IFilterItem`: See IFilterItem (Page 33).

No direct deletion, rather the FilterItem is cut out and saved temporarily.
See also Index: Start value (Page 12).

1.7.3 IFilter: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Locked

```
Locked() as Boolean
```

Locked (Page 11).

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10). Current implementations of IFilter: QFilter.

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as IQuery Read only
```

Owner (Page 11). Here: IQuery.

1.8 IFilterItem

An object that implements IFilterItem also implements IStringStorage (Page 59). Current implementations of IFilterItem: QFilterItem.

1.8.1 IFilterItem: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10).

1.8.2 IFilterItem: Function

Copy

```
Copy() as IFilterItem
```

Copy (Page 10). Here, copy of: Filter item

1.8.3 IFilterItem: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Column

```
Column() as IColumnDef
```

The column in BaseQuery to which the filter row refers. This is precisely this column.

Operator

```
Operator or() as Integer
```

The Operator of the filter row.

The Operator can process only values from qeIFilterItemOperator (Page 118). If the Query is used together with QueryBrowser, then QueryBrowser prevents the use of values other than those from qeIFilterItemOperator.

If FilterItem.Operator is addressed directly, then it is possible to enter something else (the "Integer" data type is permitted here); but any entry other than the values of qeIFilterItemOperator would produce errors.

Value

```
Value() as Variant
```

In the user interface, corresponds to "Filter value".

FType

```
FType() as qeIFilterItemType
```

FType varies depending on whether the FilterItem is an expression or whether it consists of permissible, logical expressions (And, Or, Not etc.). Default: qcExpression.

See also qeIFilterItemType (Page 113)

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10)

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as IFilter Read only
```

Owner (Page 11). Here: IFilter

CaseSensitive

```
CaseSensitive() as Boolean
```

True: Case sensitivity activated. In the user interface: corresponds to the "Case sensitive" column.

1.9 ISort

1.9.1 ISort: Sub

ShutDown

```
ShutDown() ShutDown
```

Clear

```
Clear()
```

Deletes all items, hence all sort objects.

1.9.2 ISort: Function

Count

```
Count() as Integer
```

Number of SortItems.

Item

```
Item(ByVal Index as Integer) as ISortItem
```

- `Index`: Index if SortItem.
- `ISortItem`: See section ISortItem (Page 37).

Returns the item identified by Index.
See also Index: Start value (Page 12).

AddNew

```
AddNew(Optional ByVal Index as Integer = -1) as ISortItem
```

- `Index`: Either the index where the new SortItem is to be used (adds 1 to the subsequent indices). If -1, it is appended at the end of collection.
- `ISortItem`: See section ISortItem (Page 37).

See also Default value "-1", Index: Start value (Page 12).

Delete

```
Delete(ByVal ItemIdent as Variant) as Boolean
```

- `ItemIdent`: Either the index of SortItem or SortItem object.

Deletes a SortItem.
See also Index: Start value (Page 12).

Copy

```
Copy() as ISort
```

Copy (Page 10): Here, copy of: Sort object

Add

```
Add(ByVal ItemObject as ISortItem, Optional ByVal Index as Integer =
-1) as Integer
```

- `ItemObject`: See ISortItem (Page 37).
- `Index` Either the index where the temporarily saved SortItem is to be used (adds 1 to the subsequent indices). If -1, it is appended at the end of collection.

See also Default value "-1", Index: Start value (Page 12).

Remove

```
Remove(ByVal ItemIdent as Variant) as ISortItem
```

- `ItemIdent`:Either the index of SortItem or SortItem object

No direct deletion, rather SortItem is cut out and temporarily saved.
See also Index: Start value (Page 12).

See also ISortItem (Page 37)

1.9.3 ISort: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Locked

```
Locked() as Boolean
```

Locked (Page 11).

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

See also IStringStorage (Page 59)

Owner

```
Owner() as IQuery Read only
```

Owner (Page 11). Here: IQuery

1.10 ISortItem

1.10.1 ISortItem: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10).

1.10.2 ISortItem: Function

Copy

```
Copy() as ISortItem
```

Copy (Page 10). Here, copy of: SortItem

1.10.3 ISortItem: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Column

```
Column() as IColumnDef
```

The column of the sorting amongst other has the input to which the column of the BaseQuery filter row refers. This is precisely this column.

SortOrder

```
SortOrder() as qeISortOrder
```

In the interface, corresponds to the "Sequence" column.

See also qeISortOrder (Page 114)

SortType

```
SortType() as qeISortType
```

In the interface, corresponds to the "Sort type" column.

See also qeISortType (Page 114)

GroupLevel

```
GroupLevel() as Integer
```

In the user interface: corresponds to the "Grouping" column. Default: = (No grouping). The grouping is the prerequisite for hierarchical queries. Without GroupLevel there are no subqueries.

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as ISort Read only
```

Owner (Page 11). Here: ISort

CaseSensitive

```
CaseSensitive() as Boolean True
```

: Case sensitivity activated. In the user interface: corresponds to the "Case sensitive" column.

1.11 IColumnDefs

All objects that implement the IColumnDefs, also implement IStringStorage (Page 59).

1.11.1 IColumnDefs: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10).

Clear

```
Clear()
```

Deletes all columns from the collection.

1.11.2 IColumnDefs: Function

Count

```
Count() as Integer
```

Number of columns.

Item

```
Item(ByVal ColumnIdent as Variant) as IColumnDef
```

- `ColumnIdent`: Either column index (Integer ranging from 1 to IQuery.ColumnCount) or column name (String). Thus, ColumnIdent is defined exactly as "ColIdent". Input difference: see Name (Page 42).

Returns a column, which itself is an object.

See also IColumnDef (Page 41)

AddNew

```
AddNew(ByVal Name as String, Optional ByVal Index as Integer = -1)
as IColumnDef
```

- `Name`: Name of the new column, see Name (Page 42).
- `Index`: Either the index where the column is to be inserted, or at -1, at the end of the collection. See also Default value "-1", Index: Start value. (Page 12)

See also IColumnDef (Page 41)

Delete

`Delete(ByVal ItemIdent as Variant) as Boolean`

- `ItemIdent`: Either the index of ColumnDef or the ColumnDef object.

Deletes a column.
See also Index: Start value (Page 12).

Copy

`Copy() as IColumnDefs`

Copy (Page 10).

Add

```
Add(ByVal ItemObject as IColumnDef, Optional ByVal Index as Integer
= -1) as Integer
```

- `ItemObject:` See IColumnDef (Page 41)`.`
- `Index`:: Either the index at which the temporarily saved ColumnDef is to be used (the column that was previously here gets a by 1 increased index) or end of collection.
  See also Default value "-1", Index: Start value. (Page 12)

Inserts a memorized ColumnDef.

Remove

`Remove(ByVal ItemIdent as Variant) as IColumnDef`

- `ItemIdent`: Either the index of ColumnDef or ColumnDef object

No direct deletion, rather the column is cut out and saved temporarily.
Also see Index: Start value (Page 12).

See also IColumnDef (Page 41)

1.11.3 IColumnDefs: Property

Version

`Version() as Integer Read only`

Version (Page 10).

Locked

`Locked() as Boolean`

Locked (Page 11).

IsChanged

`IsChanged() as Boolean`

IsChanged (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as IBaseQuery Read only
```

Owner (Page 11). Here: IBaseQuery.

1.12 IColumnDef

1.12.1 IColumnDef

All objects that implement the `IColumnDef` also implement IStringStorage (Page 59) and `IColumnValue`.

1.12.2 IColumnDef: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10).

1.12.3 IColumnDef: Function

Copy

```
Copy() as IColumnDef
```

Copy (Page 10). Here, copy of: ColumnDef. Transferred to the interface: In the interface, select |Options, "Column edit" tab. There, you can copy a row (one row represents exactly one column) and insert it again. This also works universally: You can copy such a with a column definition and insert it in another query.

EditableAllowed

```
EditableAllowed() as Boolean
```

Calculates whether it is permitted to change a property. Returns False in the following cases:

- Editable (Page 42) is False: not editable.
- Editable (Page 42) is True, but the property is not editable due to program-specific reasons. For example, AliasFullName can never be used independently, but is always calculated by the system.

However, the "Object per value" setting overrides EditableAllowed.

SetObjectAllowed

```
SetObjectAllowed() as Boolean
```

Calculates if an object can also be set from the outside in an object calculation, for example, in the QueryBrowser by means of drag&drop. In following cases, SetObjectAllowed returns False:

- SetObject (Page 42) is False (the object may not be set).
- SetObject (Page 42) is True, but the object cannot be set for program-specific reasons. For example, it is not permitted to set an owner; but instead is always set by the system.

The two conditions above only apply for the calculation type "Navigation library: short" and "Navigation library: extended". The "Script" calculation type bypasses this function!

1.12.4 IColumnDef: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Name

```
Name() as String
```

IColumnDef.Name has the following input checks:

- Prohibits an empty string
- Uniqueness regarding already existing names
- Rejection of a string whose first character is a number.

Background:

- The `ColIdent` and `ColumnIdent` parameters are used in various functions. These two parameters either allow the input of the column index or the column name. The user can simply transfer a variant-type variable and does not have to explicitly state whether he wants to enter the index or name. For the component to be able to automatically decide if the index or name is to be addressed, there has to be a clear distinction.

All entries that also include alphabets are automatically interpreted as names. In addition, you must also prevent the selection of a name that consists only of numbers. Otherwise, there is the possibility of mix-ups. For this purpose, all strings that begin with a number are rejected as names. This is essentially a narrower definition than would be necessary. But in this way, the input check remains efficient, as only the first character is checked and not all characters.

Note

If a column is moved, the column receives a new column index. That is why it is recommended to use the name whenever this is possible.

Description

```
Description() as String
```

Free.

DisplayDescription

```
DisplayDescription() as String Read only
```

Free. If empty: QueryBrowser displays the Name instead of DisplayDescription.

ScreenSize

```
ScreenSize() as Integer
```

Is processed together with SizeUnit. Without SizeUnit, ScreenSize is a unitless number.

SizeUnit

```
SizeUnit() as qeColumnSizeUnit
```

Is processed together with *ScreenSize*.

See also qeColumnSizeUnit (Page 112)

Visible

```
Visible() as Boolean
```

Corresponds in the interface to: Properties of the column, "General" tab: Visible. An invisible column continues to be available via the column name or index. The visibility or invisibility does not affect the column index: The index is always calculated across all columns.

withPicture

```
withPicture() as Boolean
```

Defines whether Picture (Page 54) will be evaluated.

- If an Extender IExtended (Page 61) exists, the icon is returned from there.
- If there is no Extender but at least one COMOS object: The standard icon of the COMOS object is returned.

ScriptTextFunctionValue

`ScriptTextFunctionValue() as String`

Returns the `script` from the following location: Properties of a column, Value calculation tab, `Script` calculation type. This script can be used to calculate a cell value.

Note

`ScriptTextFunctionValue` returns only the function, without the `Function` header and without the `EndFunction` footer.

ScriptTextFunctionObject

`ScriptTextFunctionObject() as String`

Returns the script from the following location: Properties of a column, Object evaluation tab, Script calculation type. This script can be used to calculate a cell object.

Note

`ScriptTextFunctionObject` returns only the function, without the `Function` header and without the `EndFunction` footer.

DependOf

`DependOf() as IColumnDef`

Defines the reference object from which the calculation is done. In the interface, this is the Reference field in the "General" tab.

Tag

`Tag() as Variant`

For internal COMOS purposes. For example, is used for queries that use the SystemExtender.

ShowProperty

`ShowProperty() as qeColumnDefProp`

Returns standard properties.

In the user interface corresponds to the Display list in the Value calculation tab, Standard property calculation type.

The list only offers a selection of frequently required properties. This list does not claim to be complete or that a property also supports a specific individual purpose. The user can also determine his own properties by using a script or an expression.

`ShowProperty` requires an input of *Parameter*.
See also qeColumnDefProp (Page 111)

Parameter

`Parameter() as String`

Is used by *ShowProperty* . Returns the parameter that is used to determine a property.

Sometimes, the parameter is set by the code. In this case, *RelativeObject* is used.

NavigationObject

```
NavigationObject() as ObjNavigator
```

The ObjNavigator object type comes from ocx\ ObjNavigator.dll.

ObjNavigator calculates and manages the navigation steps. Hence, you can consider it as a collection of steps. ObjNavigator is never Nothing; but the instance always exists.

Numeric

```
Numeric() as Boolean
```

Was used by ISort (Page 35) in the past. Is outdated today and is replaced by *DefaultValueType* .

DefaultValueType

```
DefaultValueType() as qeValueType
```

Checks that each entry has a type.

This property only determines that each entry in the query environment has at least one ValueType. Then, if the value is processed, for example in the QueryBrowser, it is permissible to assign another ValueType there.

- In attributes, a numeric value is set as default. In other words, each value coming from an attribute is initially interpreted as numerical (Double).
- Default for all other properties of COMOS objects: Entries are initially interpreted as string.
- Especially for attribute type Date: Input is interpreted as data type Date in the default.

See also qeValueType (Page 116)

Locked

```
Locked() as Boolean
```

Locked (Page 11).

FilterEnabled

```
FilterEnabled() as Boolean
```

Corresponds in the interface to: Filter option allowed in the "General" tab. This option controls if the user may define his own filter.

SortEnabled

```
SortEnabled() as Boolean
```

Corresponds in the interface to: Sorting option allowed in the "General" tab. This option controls whether the user may define an own sorting.

GroupEnabled

`GroupEnabled() as Boolean`

Corresponds in the interface to: Grouping option allowed in the General tab. The grouping is controlled via the sorting.

RelativeObject

`RelativeObject() as Object`

Is used by *Parameter* .
Example: `AliasRelativeLabel` sets the object from the "Start object" field (corresponds to `TopQuery.MainObject`) as `Parameter`. In this case, the user cannot set the start object in the user interface. However, the `Parameter` property remains accessible via code.

IsChanged

`IsChanged() as Boolean`

IsChanged (Page 10).

Dispatch

`Dispatch() as Object Read only`

Dispatch (Page 10).

ExtendedType

`ExtendedType() as Long`

Here, a sum of the constants of qeColumnExtendedType (Page 118) can be formed. The sum specifies which of the options is calculated by IExtended (Page 61). You can either select one or all of the options, and the constants will be added up. In the code, you can conclude from the sum which options were selected.

ComosPhysUnitName

`ComosPhysUnitName() as String`

Here the Comos unit is managed in the case that a column contains Comos attributes.

In Comos, each attribute is managed with a Value and a PhysUnit. If you include the attributes into the object query, then you can implement the PhysUnit of the values and thereby receive other DisplayValues.

Example:

Values are entered as "m" in the attributes. The attributes are included into the query, but there are displayed with the PhysUnit "mm". Obviously, the column will display another DisplayValue, which, when considering the PhysUnit, will have the same value as the attribute in the tab.

Please note: The above mechanism works quite intuitively if both the Value and the PhysUnit are simultaneously displayed in a column. Misunderstandings can never arise in such a case.

But there is also the possibility to display Value and Unit in two different columns. Unit is not identical with PhysUnit. In this case, Value is still converted using PhysUnit. However, Unit is retained – and would now lead to a wrong interpretation of DisplayValue.

Alignment

```
Alignment() as qeColumnAlignment
```

Alignment within the cell. In the user interface: corresponds to the column properties, Extra tab, Alignment option.
See also qeColumnAlignment (Page 111)

WrapText

```
WrapText() as Boolean
```

Word-wrap within the cell. In the user interface: corresponds to the column properties, Extra tab, Text wrap at cell border option.

SizeByUnit

```
SizeByUnit(ByVal CalcUnit as qeColumnSizeUnit) as Long
```

- `CalcUnit`: See qeColumnSizeUnit (Page 112).

The conversion of the ScreenSize. Possible conversion units: mm, twips, multiple of row height.

Editable

```
Editable() as Boolean
```

Allows the editing of the cell. In the user interface, corresponds to the column properties, Value calculation tab, Editable option.

IsSum

```
IsSum() as Boolean
```

Calculates the sum and displays it in a sum field at the bottom of the column.

Charset

```
Charset() as Integer
```

Object queries support all CharSets that are also supported by VB6. Please note: Depending on the concrete implementation, there can be restrictions when using CharSets, because the object query of other components is limited. For example, the Comos.dll does not support the CharSets as universally as the object queries.

LCID

```
LCID() as Long
```

International standard for labeling languages. Particularly important for import/export of languages.

InternationalDescription

```
InternationalDescription(ByVal Index as Integer) as String
```

- `Index`: Language index

The object queries save the translations as a pair from *LCID* and this InternationalDescription. Thus, in the object queries – unlike in the language management in Comos – the number and sequence of languages is arbitrary.

See also Index: Start value (Page 12).

ExtendedProgID

```
ExtendedProgID() as String
```

Normally, the ExtendedProgID will be identical for all columns. Technically, however, it is possible to manage various ExtendedProgID.

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

Owner

```
Owner() as IColumnDefs Read only
```

Owner (Page 11). Here: IColumnDefs

Eval

```
Eval() as IColumnEval Read only
```

Switches to IColumnEval (Page 51).

PreLoad

```
PreLoad() as Boolean
```

Enables special management of attributes. Calls the PreLoadSpecs function from the project.

ScriptTextFunctionText

```
ScriptTextFunctionText() as String
```

Outdated, but is still available for compatibility reasons. Please use *ScriptTextFunctionValue* instead.

SetObject

```
SetObject() as Boolean
```

Corresponds to the checkbox in the Object evaluation tab: Object can be set.

Is controlled by *SetObjectAllowed*: Only if `SetObjectAllowed` is true, you can also set `SetObject` to true.

ScriptTextFunctionObjectByValue

```
ScriptTextFunctionObjectByValue() as String
```

Corresponds to the script text that is entered on the Object by value tab in the user interface.

Note

`ScriptTextFunctionObjectByValue` returns only the function body, without the `Function` header and without the `EndFunction` footer.

IsKeyColumn

```
IsKeyColumn() as Boolean
```

Is used only when importing XML connectors. You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword ""Key column".

IsMaster

```
IsMaster() as Boolean
```

Is used when the query generates new objects. The object of this column (= cell) is applied in the OrigCollection (Page 17).

See also qeEvalByValueType (Page 114).

MappingInfo

```
MappingInfo() as String
```

Outdated, only remains existent due to compatibility reasons.
Better: ColumnDef.Mapping.MappingInfo.

ObjectByValueOwner

```
ObjectByValueOwner() as IRunObjectDef Read only
```

Corresponds in the interface to: Object by value tab, calculation type "Default definitions (engineering objects)", field: Owner object.

See also IRunObjectDef (Page 50)

ObjectByValueCDevice

```
ObjectByValueCDevice() as IRunObjectDef Read only
```

Corresponds in the interface to: Object by value tab, calculation type "Default definitions (engineering objects)", field: Base object.

See also IRunObjectDef (Page 50)

CalculateType

```
CalculateType() as qeCalculateType Read only
```

See Enum.

See also qeCalculateType (Page 116)

ValueItemsSortType

`ValueItemsSortType() as qeVItemsSortType`

This property can be used only if the cell contains a list (dropdown menu). In this case, you can define which sorting this list should have.

See also qeVItemsSortType (Page 113)

Mapping

`Mapping() as IMapping Read only`

Mapping for the XML connectors.

See also IMapping (Page 94)

ObjectByValueStdOptions

`ObjectByValueStdOptions() as Long`

Corresponds in the interface to: Object by value tab, calculation type "Default definitions (engineering objects)", checkbox: "With sorting into sublevels".

Refers to Enum qeObjectByValueStdOptions (Page 118).

1.13 IRunObjectDef

1.13.1 IRunObjectDef: Property

Version

`Version() as Integer Read only`

Version (Page 10).

SType

`SType() as qeRunObjectType`

Corresponds in the interface to: "Object by value" tab, "Default" calculation type, Field: "Base object". The Enum determines the options specified in the dropdown list. See also Parameter.
See also qeRunObjectType (Page 116)

Parameter

`Parameter() as Variant`

Additional parameters depending on the selection for SType.

- Example:
  Selection of the user-defined object option. Use drag&drop to set an object; the PathFullName of this object is determined.
- Example:
  Object of the neighbor cell (= Name of neighbor column).
- Example:
  SystemFullName: Drag&drop an object, from which the SystemFullName is determined.

1.14 IColumnEval

Here the IColumnDef (Page 41) and IRunObjectDef (Page 50) settings are used.

1.14.1 IColumnEval: Function

CheckSetEvalObject

```
CheckSetEvalObject(ByVal MainRowObject as Object, ByVal RefColObject
as Object, ByVal vNewValue as Object) as Boolean
```

Refers to the "Object settable" option. The function checks if you may set the object.

- `MainRowObject`: Row object of OrigCollection (Page 17).
- `RefColObject`: Object of the reference column.
- `vNewValue`: The object that is to be checked (the one that is to be set).

Also calls CheckSetCellObject (Page 63).

SelectEvalObject

```
SelectEvalObject(ByRef Cancel as Boolean, ByVal MainRowObject as
Object, Optional ByVal RefColObject as Object) as Object
```

Under certain conditions, you may assign an object to a cell by clicking into the cell (three points are visible in the cell) and an additional browser opens.

- `Cancel`: Cancels the additionally opened window.
- `MainRowObject`: Row object of OrigCollection (Page 17).
- `RefColObject:` Object of the reference column.

Requires that SetObjectAllowed (Page 42) or SetObject (Page 42) = True is set. ?

SelectEvalValue

```
SelectEvalValue(ByRef Cancel as Boolean, ByVal MainRowObject as
Object, Optional ByVal ColumnObject as Variant) as Variant
```

Under certain conditions, you may assign a value to a cell by clicking into the cell (three points are visible in the cell) so that an additional browser will open.

- `Cancel`: Cancels the additionally opened window.
- `MainRowObject`: Row object of OrigCollection (Page 17).
- `ColumnObject`: Current object in the cell.

1.14.2 IColumnEval: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

EvalObject

```
EvalObject(ByVal MainRowObject as Object, Optional ByVal
RefColObject as Object) as Object
```

Runtime object of cell.

- `Get`: Calculates the object of the cell. Always allowed.
- `Let/Set`: Only if SetObjectAllowed (Page 41) SetObject at ColumnDef is set to True. SetObjectAllowed (Page 41) or SetObject (Page 42)?
- `MainRowObject`: Row object.
- `RefColObject`: Reference object of the column.
- `Output`: ColumnObject.

EvalValue

```
EvalValue(ByVal MainRowObject as Object, Optional ByVal ColumnObject
as Variant) as Variant
```

Runtime value of the cell.

Takes ColumnObject from EvalObject and determines the to be displayed value.

If a typified value can be determined (e.g. a date), then it will be maintained here as well, therefore Variant.

- `Let`: If EditableAllowed (Page 41) or Editable (Page 42) = True at ColumnDef, then you may make entries here. ?
- `MainRowObject`: Row object.
- `ColumnObject`: Current object in the cell.

ValueItems

```
ValueItems(ByVal MainRowObject as Object, Optional ByVal
ColumnObject as Variant) as IValueItems Read only.
```

These are the values of a dropdown list.

- `MainRowObject`: Row object.
- `ColumnObject`: Current object in the cell.

See also IValueItems (Page 56)

EvalTypedValue

```
EvalTypedValue(ByVal MainRowObject as Object, Optional ByVal
ColumnObject as Variant, Optional ByVal Value as Variant, Optional
ByVal ValueType as qeValueType = 12) as Variant Read only.
```

This belongs to *EvalValue*. Here a typified value is converted into another typified value. Example: Convert Date to a string that still looks like a Date but no longer is one.

Supported types: See Enum qeValueType (Page 116).

- `MainRowObject`: Row object.
- `ColumnObject`: Current object in the cell.

EvalPicture

```
EvalPicture(ByVal MainRowObject as Object, Optional ByVal
ColumnObject as Variant, Optional ByVal PictureType as qePictureType
= 0) as Picture Read only.
```

Similar to *EvalValue*: here the picture is calculated.

- `MainRowObject`: Row object.
- `ColumnObject`: Current object in the cell.

See also qePictureType (Page 115)

EvalStyle

```
EvalStyle(ByVal MainRowObject as Object, Optional ByVal ColumnObject
as Variant) as IStyle Read only.
```

Similar to *EvalValue*: the Style is calculated here.

- `MainRowObject`: Row object.
- `ColumnObject`: Current object in the cell.

See also IStyle (Page 59)

EvalObjectByValue

```
EvalObjectByValue(ByVal MainRowObject as Object, ByVal ColObject as
Object, ByVal vNewValue as Variant, ByRef IsValid as Boolean) as
Object Read only.
```

The standard case in an object query is `EvalObject`. In addition, you can use this property.

Corresponds in the interface to: the "Object by value" tab.

If you are using `EvalObjectByValue`, *EvalObject* will be called in addition, to check whether `EvalObjectByValue` returns the "correct" object.

If `EvalObjectByValue` and the subsequently called *EvalObject* differ, then the check routine returns `False`.

1.15 ICell

1.15.1 ICell: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10).

1.15.2 ICell: Function

EvalByValue

```
EvalByValue() as Boolean
```

EvalByValue of the cell. Is called by EvalByValue (Page 20) or `EvalByValue` of the row. ?

1.15.3 ICell: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Object

```
Object() as Object
```

This is the object that was computed by IColumnEval (Page 51).

Text

```
Text() as String
```

This is the text that was computed by IColumnEval (Page 51).

If you query `Cell.Text`, then `Cell.Value` is first computed, and only then Cell.Text.

Picture

`Picture() as Picture Read only.`

This is the picture that was computed by IColumnEval (Page 51).

RowIndex

`RowIndex() as Long Read only.`

Begins at 1.

This is the `BaseRowIndex`, as the cell is only available at IBaseQuery (Page 26).

ColIndex

`ColIndex() as Integer Read only.`

Begins at 1.

This is the `BaseColIndex`, as the column is only available at IBaseQuery (Page 26).

Tag

`Tag() as Variant`

Internal use.

IsChanged

`IsChanged() as Boolean`

IsChanged (Page 10). Here, check for change of: `Cell`.

Dispatch

`Dispatch() as Object Read only.`

Dispatch (Page 10).

Style

`Style() as IStyle`

Returns what was IColumnEval (Page 51) computed in `EvalStyle`.

Owner

`Owner() as IBaseQuery read only`

Owner (Page 11). Here: IBaseQuery (Page 26)

NumericValue

`NumericValue() as Double Read only.`

Calculated by IColumnValue.

If you query `Cell.NumericValue`, `Cell.Value` is calculated first and only then `Cell.NumericValue`.

ValueItems

```
ValueItems() as IValueItems
```

Returns what was computed by IColumnEval (Page 51).

`IColumnEval` is calculated by EvalValueItems (Page 52).

Value

```
Value() as Variant
```

Before you can query the Value, you must have computed `Cell.Object`. And before you can compute `Cell.Object`, you must compute the reference column.

IsValid

```
IsValid() as Boolean Read only.
```

IsValid of the cell. Is called by IsValid (Page 23) of the row.

Relevant for the `ObjectByValue` function.

Permissions

```
Permissions() as qePermissions Read only.
```

See Enum.
See also qePermissions (Page 113)

1.16 IValueItems

Returns the selection options for the values.
Example:
The results that are displayed in a dropdown menu.

1.16.1 IValueItems: Sub

Sort

```
Sort()
```

Sorts the ValueItem.

1.16.2 IValueItems: Function

Add

```
Add(ByVal VItem as IValueItem, Optional ByVal Index as Long = -1) as
Long
```

Adds a `ValueItem`; either at the end (-1) or at the position indicated by the `Index`.

- `Index`: Position where the item is to be inserted.
  See also Default value "-1", Index: Start value (Page 12).

See also IValueItem (Page 58)

Remove

```
Remove(ByVal ItemIdent as Variant) as IValueItem
```

Removes a `ValueItem` from the listing.

- `ItemIdent`: Identifies the to be removed item. `ValueItem` is identified by index or object. There is no identification by name here.

See also Index: Start value (Page 12).

AddNew

```
AddNew(ByVal Value as Variant, ByVal DisplayValue as Variant,
Optional ByVal Index as Integer = -1) as IValueItem
```

`ValueItem` consists of `Value` and `DisplayValue`.

- `Value`: Value
- `DisplayValue`: Displayed text
- `Index`: Position where the item is to be inserted.
  See also Default value "-1", Index: Start value (Page 12).

See also IValueItem (Page 58)

1.16.3 IValueItems: Property

Count

```
Count() as Long Read only.
```

Number of ValueItems.

Item

```
Item(ByVal Index as Long ) as IValueItem Read only.
```

- `Index` Returns the `ValueItem` (conceptually, `ValueItem` and Item are the same here).

See also Index: Start value (Page 12).

Version

`Version() as Integer Read only`

Version (Page 10).

Presentation

`Presentation() as qePresentation`

Controls the presentation, see Enum.

See also qePresentation (Page 112)

SortType

`SortType() as qeVItemsSortType`

See Enum.

See also qeVItemsSortType (Page 113)

1.17 IValueItem

1.17.1 IValueItem: Property

Value

`Value() as Variant`

The value of a cell in the object query.

DisplayValue

`DisplayValue() as Variant`

The displayed cell value in the object query. The `DisplayValue` corresponds to `Query.Cell(i,j).Text`.

Version

`Version() as Integer Read only`

Version (Page 10).

1.18 IStyle

1.18.1 IStyle: Property

BackColor

`BackColor() as Long`

The cell background color.

ForeColor

`ForeColor() as Long`

The font color.

Font

`Font() as Font`

Font: Typ StdOLE.IFont.

Version

`Version() as Integer Read only`

Version (Page 10).

TipText

`TipText() as String`

The tooltip of the cell.

1.19 IStringStorage

Serialization of query objects in XML (based on DOM1). Serialization is the process of saving an object in order to restore the object later on. The items of the Childs collection are evaluated recursively during this process.

1.19.1 IStringStorage: Sub

Load

`Load(ByRef CurrentNode as IXMLDOMNode)`

Deserialization: Loads the settings, variables etc. in order to restore the object.

- `IXMLDOMNode`: See the explanation in Save (Page 101).

1.19.2 IStringStorage: Function

Save

`Save(ByVal OwnerNode as IXMLDOMNode, ByVal CurrentNode as IXMLDOMNode) as IXMLDOMNode`

Serialization (saving the object data).

- `IXMLDOMNode`: See the explanation in Save (Page 101).

CurrentStringStorage

`CurrentStringStorage() as String`

Serialization in XML format, but as XML string (and not as DOM model).

1.19.3 IStringStorage: Property

Version

`Version() as Integer Read only`

Version (Page 10).

Locked

`Locked() as Boolean`

Locked (Page 11).

IsChanged

`IsChanged() as Boolean`

- IsChanged (Page 10). Here, check for change of:
  - `StringStorage`

ArchivIdent

`ArchivIdent() as String Read only.`

Version number of the serialization structure. If the properties, variables etc. of an object change, then the volume and structure of the information that is saved in the serialization changes accordingly.

Name

`Name() as String Read only.`

Unique key of the to be serialized instance. This key clearly identifies the object when multiple instances of the same data type exist.
Example:

When serializing a query tehre are multiple columns which have to be differentiated. For columns, you take the column name.

InheritMode

```
InheritMode() as qeInheritMode
```

For internal purposes.

See also qeInheritMode (Page 112)

1.20 IExtended

Query extension for free use.

1.20.1 IExtended: Sub

Definitions

```
Definitions(ByVal BaseQuery as IBaseQuery )
```

Is called by IBaseQuery (Page 26). Here you can define your own columns or change existing columns.

TQBrowserDefinitions

```
TQBrowserDefinitions(ByVal TQBrowser as ITopQueryBrowser)
```

Is called by ITopQueryBrowser (Page 65). Processes the events of ITopQuery (Page 17) and ITopQueryBrowser.

Note

These events are especially intended there. No Windows events are routed through.

ShutDown

```
ShutDown()
```

ShutDown (Page 10).

CellStyle

```
CellStyle ( ByVal MainRowObject as Object, ByVal ColumnDef as
IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer,
ByVal CellObject as Object, ByRef Style as IStyle )
```

Is only called by the system if the ExtendedType (Page 42) is used at IColumnDef (Page 41).

- `MainRowObject`: Row object.
- `ColumnDef`: See there.
- `RowIndex`: See RowIndex (Page 54).
- `ColumnIndex`: Corresponds to ColIndex.
- `CellObject`: Cell object. The cell object must already have been evaluated at this point.
- `Style`: Style object. The developer must parameterize the style object here.

See also Index: Start value (Page 12).

See also IStyle (Page 59)

GetOrigCollection

```
GetOrigCollection(ByVal TopQuery as ITopQuery, ByRef
ObjectsCollection as Object)
```

`TopQuery` calls the Extender and returns the `ObjectsCollection` (this concerns the OrigCollection (Page 17)).

Normally, the `TopQuery` returns the `OrigCollection` to `BaseQuery`. But the Extender can still intervene exactly here: the Extender accepts `OrigCollection` from `TopQuery`, changes it and instead of the `OrigCollection` returns this `ObjectsCollection`.

See also OrgCollectionByExtendedOnly (Page 64).

See also ITopQuery (Page 17)

CellValueItems

```
CellValueItems(ByVal MainRowObject as Object, ByVal ColumnDef as
IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer,
ByVal CellObject as Object, ByRef ValueItems as IValueItems)
```

Is only called by the system if the ExtendedType (Page 42) is used at IColumnDef (Page 41).

- `MainRowObject`: Row object.
- `ColumnDef`: See there.
- `RowIndex`: See RowIndex (Page 54).
- `ColumnIndex`: Corresponds to ColIndex.
- `CellObject`: Cell object. The cell object must already have been evaluated at this point.
- `ValueItems`: ?

See also Index: Start value (Page 12).

SUIQueryDefinitions

```
SUIQueryDefinitions(ByVal Specification as IComosDSpecification,
ByVal Query as IQuery, ByVal QueryBrowser as IQueryBrowser)
```

`SUIQueryDefinition` is called if the query runs under `SUIQuery`.

This is an alternative option to running the query under `TopQueryBrowser`. If the query can be used in the Comos user interface, only one of the two can be active: `TopQueryBrowser` or `SUIQuery`, but never both.

Consequence:
Without an interface, neither `SUIQueryDefinitions` nor `TopQueryBrowserDefinitions` can be triggered.

See also IQuery (Page 20)

See also IQueryBrowser (Page 72)

1.20.2 IExtended: Function

CellObjectOld

```
CellObjectOld(ByVal MainRowObject as Object, ByVal ColumnDef as
IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer) as
Object
```

For compatibility reasons. Is needed only for `qcIExtended.Version` <6.

See also IColumnDef (Page 41)

CellPicture

```
CellPicture(ByVal MainRowObject as Object, ByVal ColumnDef as
IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer,
ByVal CellObject as Object) as Picture
```

Is only called by the system if the ExtendedType (Page 42) is used at IColumnDef (Page 41).

- `MainRowObject`: Row object.
- `ColumnDef`: See there.
- `RowIndex`: See RowIndex (Page 54).
- `ColumnIndex`: Corresponds to ColIndex.
- `CellObject`: Cell object. The cell object must already have been evaluated at this point.
- `Picture`: The picture of the cell. The developer must make sure that the picture is returned here.

See also Index: Start value (Page 12).

Reasonable

```
Reasonable(ByVal ItemObject as Object) as Boolean
```

Is derived from `TopQueryReasonable`. Only for internal use.

CheckSetCellObject

```
CheckSetCellObject(ByVal MainRowObject as Object, ByVal ColumnDef as
IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer,
ByVal RefColObject as Object, ByVal vNewValue as Object) as Boolean
```

- `MainRowObject`: Row object.
- `ColumnDef`: See there.
- `RowIndex`: See RowIndex (Page 54).
- `ColumnIndex`: Corresponds to `ColIndex`.
- `RefColObject`: ?
- `vNewValue`: ?

Is called by CheckSetEvalObject (Page 51).

See also Index: Start value (Page 12).

See also IColumnDef (Page 41)

1.20.3 IExtended: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

CellValue

```
CellValue(ByVal MainRowObject as Object, ByVal ColumnDef as
IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer,
ByVal CellObject as Object) as Variant
```

- `MainRowObject`: Row object.
- `ColumnDef`: See there.
- `RowIndex`: See RowIndex (Page 54).
- `ColumnIndex`: Corresponds to ColIndex.
- `CellObject`: Cell object. The cell object must already have been evaluated at this point.

`Get` and `Let` are allowed.

See also Index: Start value (Page 12).

See also IColumnDef (Page 41)

OrgCollectionByExtendedOnly

```
OrgCollectionByExtendedOnly() as Boolean Read only.
```

- True TopQuery does not return a `OrigCollection`; the Extender returns an `ObjectsCollection` and passes it to `BaseQuery` (see the explanation in GetOrigCollection (Page 61)). This saves time.

CellObject

`CellObject(ByVal MainRowObject as Object, ByVal ColumnDef as IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer, ByVal RefColObject as Object) as Object`

- `MainRowObject`: Row object.
- `ColumnDef`: See there.
- `RowIndex`: See RowIndex (Page 54).
- `ColumnIndex`: Corresponds to ColIndex.
- `RefColObject`: ?

`Get` and `Let` are allowed.

See also Index: Start value (Page 12).

See also IColumnDef (Page 41)

NewMainRowObject

`NewMainRowObject() as Object Read only.`

The Extender sets the `MainRowObject` for `BaseQuery`. Called by *IBaseQuery.* CreateNewRow (Page 27).

CellObjectByValue

`CellObjectByValue(ByVal MainRowObject as Object, ByVal ColumnDef as IColumnDef, ByVal RowIndex as Long, ByVal ColumnIndex as Integer, ByVal CellObject as Object, ByVal vNewValue as Variant, ByRef IsValid as Boolean) as Object Read only.`

Called by `ColumnObjectByValue` (see Style above). The `ColumnObjectByValue` script is called in queries and is used to create new objects.

See also Index: Start value (Page 12).

See also IColumnDef (Page 41)

Dispatch

`Dispatch() as Object Read only`

Dispatch (Page 10).

1.21 ITopQueryBrowser

The user interface of Implementations of ITopQuery (Page 17), as long as it pertains to the part applicable for all.

1.21.1 ITopQueryBrowser: Sub

ShutDown

```
ShutDown()
```

ShutDown.

EventsHandlerAdd

```
EventsHandlerAdd(ByVal vNewValue as ITopQBrowserEvents)
```

- `vNewValue`: The new event that is to be registered.

Note

`TopQueryBrowser` (here) and `QueryBrowser` (further below) have different events.

The operating system events are not made available in `TopQueryBrowser`. Thus, which events are permitted has nothing to do with the Windows `MessageQue` or with the VB events.

The list of available events can be found in ITopQBrowserEvents (Page 69). These, and only these, are the events that can be used in `TopQuery`.

In `EventsHandlerAdd` it is declared that an event is possible. Then, in `ITopQBrowserEvents` , it is implemented what to do with this event.

EventsHandlerRemove

```
EventsHandlerRemove(ByVal vNewValue as ITopQBrowserEvents)
```

- `vNewValue`: The event to deregister.

Otherwise see *EventsHandlerAdd*.

RefreshByItems

```
RefreshByItems()
```

Items are controls that are allowed in the browser list, i.e. the buttons, dropdown menus etc. allowed in the table. `RefreshByItems` accesses Items (Page 67).

1.21.2 ITopQueryBrowser: Function

Copy

```
Copy() as ITopQueryBrowser
```

Copy (Page 10).

Copy cannot be used meaningfully at this point because `TrueDBGrid` is not allowed to copy user interfaces. Therefore, Copy is available only for internal purposes.

1.21.3 ITopQueryBrowser: Property

Version

`Version() as Integer Read only`

Version (Page 10).

TopQuery

`TopQuery() as ITopQuery`

Registers the TopQuery at the TopQueryBrowser. There is no counterpart, that is why TopQuery does not know TopQueryBrowser.

See also ITopQuery (Page 17)

XObjContainer

`XObjContainer() as Object`

This is the Comos object that holds the current XObj, i.e. a Device or CDevice.

IsChanged

`IsChanged() as Boolean`

IsChanged (Page 10).

Here it is cheked if the TopQueryBrowser was changes, hence the settings and so on. A simple change of the displayed data does not trigger an IsChanged.

Dispatch

`Dispatch() as Object Read only.`

Dispatch (Page 10).

QueryBrowser

`QueryBrowser() as IQueryBrowser Read only.`

Registers the `QueryBrowser`. The `TopQueryBrowser` knows the `QueryBrowser`. As a consequence, the properties of the `QueryBrowser` are available here.

However there is no counterpart `QueryBrowser` does not know the `TopQueryBrowser`.

See also IQueryBrowser (Page 72)

Partners

`Partners() as IPartners Read only.`

Partners are other Comos components that can communicate with `TopQueryBrowser`. At present, these are:

1. Bulk processing
2. Product data selection

The partners can supply their setting via `IStringStorage`.

See also IPartners (Page 86)

Storage

`Storage() as IStringStorage Read only.`

Storage (Page 11).

Manages own settings (i.e. the `TopQueryBrowser` settings), and, in addition to this, also the settings of *Partners*.

See also IStringStorage (Page 59)

WorkSet

`WorkSet() as IComosDWorkset`

The Comos workset is returned upon starting Comos. See also the class documentation *comos_dll.pdf*.

ToolBarButtons

`ToolBarButtons() as IButtons Read only.`

This the TopQueryBrowser toolbar. What the buttons look like and what operating options are available is not freely disposable because MS technology is used. IButtons is a Microsoft interface and hence will be different in Vista.

Items

`Items() as ITQBItems Read only.`

Is used by RefreshByItems (Page 66). "Items" are the browser controls; in plainspeak, every icon button, dropdown menu etc. is an "Item". All controls offered on the "Input layout" tab of the query options are allowed.

See also ITQBItems (Page 88)

PreLoadMonitor

`PreLoadMonitor() as Boolean`

Is no longer supported by the Comos core as of now. In the past, this sub was used when attempting to directly load the objects in an unchanged scan procedure instead of waiting until a user interaction forced the loading.

1.22 ITopQBrowserEvents

This class, among other things, takes over the QueryBrowser events. This is done only for simplification: in this way, you can simply use TopQueryBrowser and have all necessary events.

1.22.1 ITopQBrowserEvents: Sub

Change

```
Change()
```

Is triggered when the settings have changed.

SelectionChange

```
SelectionChange()
```

Is triggered when the selection in the browser is changed. If QueryBrowser.Change is triggered, then SelectionChange is triggered as well.

ToolBarButtonClick

```
ToolBarButtonClick(ByVal Button as Object)
```

- `Button`: The clicked button.

Event on clicking the button.

ToolbarButtonMenuClick

```
ToolbarButtonMenuClick(ByVal ButtonMenu as Object)
```

Like *ToolBarButtonClick*, but here for the icon buttons that have a dropdown menu.

BeforeNewScan

```
BeforeNewScan()
```

Event before rescanning data in Comos.

Background: Such a scan returns the OrigCollection (Page 17). However, only the event is relevant at this point, not the OrigCollection.

AfterScan

```
AfterScan(ByVal ObjectsCollection as Object)
```

- ObjectsCollection: VB or Comos collection.

Event that is triggered after the data were rescanned in Comos.

OLECompleteDrag

```
OLECompleteDrag(ByRef Effect as Long)
```

QueryBrowser passes the event to this location.

See *OLEDragDrop*.

OLEDragDrop

```
OLEDragDrop(ByVal Data as Object, ByRef Effect as Long, ByRef Button
as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y as Long)
```

QueryBrowser passes the event to this location.

See *OLEDragDrop*.

OLEDragOver

```
OLEDragOver(ByVal Data as Object, ByRef Effect as Long, ByRef Button
as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y as Long,
ByRef State as Integer)
```

QueryBrowser passes the event to this location.

See *OLEDragDrop*.

OLEStartDrag

```
OLEStartDrag(ByVal Data as Object, ByRef AllowedEffects as Long)
```

QueryBrowser passes the event to this location.

See *OLEDragDrop*.

OLETitleDragDrop

```
OLETitleDragDrop(ByVal Data as Object, ByRef Effect as Long, ByRef
Button as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y
as Long)
```

QueryBrowser passes the event to this location.

See *OLEDragDrop*.

OLETitleDragOver

```
OLETitleDragOver(ByVal Data as Object, ByRef Effect as Long, ByRef
Button as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y
as Long, ByRef State as Integer)
```

QueryBrowser passes the event to this location.

See *OLEDragDrop*.

DblClick

```
DblClick(ByRef IsValid as Boolean)
```

QueryBrowser passes the event to this location.

See *DblClick*.

PopUp

```
PopUp(ByVal PopUpMenu as Object, ByVal DeadArea as Boolean)
```

QueryBrowser passes the event to this location.

See *PopUp*.

PopUpSubMenu

```
PopUpSubMenu(ByVal PopUpMenu as Object, ByVal ID as String)
```

QueryBrowser passes the event to this location.

See *PopUpSubMenu*.

PopUpCmd

```
PopUpCmd(ByVal PopUpMenu as Object, ByVal ID as String)
```

QueryBrowser passes the event to this location.

See *PopUpCmd*.

TitlePopUp

```
TitlePopUp(ByVal PopUpMenu as Object, ByVal ColumnDefI as
IColumnDef)
```

QueryBrowser passes the event to this location.

See *TitlePopUp*.

See also IColumnDef (Page 41)

TitlePopUpSubMenu

```
TitlePopUpSubMenu(ByVal PopUpMenu as Object, ByVal ID as String,
ByVal ColumnDefI as IColumnDef)
```

QueryBrowser passes the event to this location.

See *TitlePopUpSubMenu*.

See also IColumnDef (Page 41)

TitlePopUpCmd

```
TitlePopUpCmd(ByVal PopUpMenu as Object, ByVal ID as String, ByVal
ColumnDefI as IColumnDef)
```

QueryBrowser passes the event to this location.

See *TitlePopUpCmd*.

DataChange

```
DataChange()
```

Event, when data was changed in the query.

1.22.2 ITopQBrowserEvents: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

1.23 IQueryBrowser

Containing

This technology identifies a query or parts of a QueryBrowsers by means of coordinates. The object or the object index found for the coordinates is returned. Used in:

- ContainingQuery (Page 75) ,
- ContainingQueryRowIndex (Page 75) ,
- ContainingQueryColumnIndex (Page 75) ,
- ContainingObject (Page 75) ,
- ContainingColumn (Page 75).

X, Y: coordinates

Usually, the coordinates are returned by a drag&drop operation. But it is also possible to set the coordinates yourself. (find the query of position x,y).

Containing is available only in the implementation (IQueryBrowser instance). This means that the relevant QueryBrowser is already known when Containing is generated.

1.23.1 IQueryBrowser: Sub

Refresh

```
Refresh()
```

Updates the QueryBrowser. See Refresh (Page 9).

RefreshRows

```
RefreshRows()
```

Updates the rows. See Refresh (Page 9).

RefreshSelectedRows

`RefreshSelectedRows()`

Updates only the selected rows. See Refresh (Page 9).

RefreshColumns

`RefreshColumns()`

Updates the columns.

Note

The rows are refreshed as well because you cannot seperate here. Reason: the queries have row objects, but no column objects. Thus, when refreshing a column, this inevitably means that the row object is refreshed as well, and hence the entire row. See Refresh (Page 9).

ShutDown

`ShutDown()`

ShutDown (Page 10).

EventsHandlerAdd

`EventsHandlerAdd(ByVal vNewValue as IQBrowserEvents)`

- `vNewValue`: The new event that is to be registered.

  Note

  TopQueryBrowser (above) and QueryBrowser (here) have different events.

  Thus, which events are permitted has nothing to do with the Windows MessageQue or with the VB events.

The list of available events is found in IQBrowserEvents. These, and only these, are the events that can be used in TopQuery.

In EventsHandlerAdd it is declared that an event is possible. Then, in IQBrowserEvents, it is implemented what to do with this event.

EventsHandlerRemove

`EventsHandlerRemove(ByVal vNewValue as IQBrowserEvents)`

- `vNewValue`: The event to deregister. Otherwise see *EventsHandlerAdd*.

Function pair PrintData, PrintPreview and PageSetup

Restricted functional scope, since external components are called here. Therewith you can print the list yourself, as if you would print an Excel list yourself.

PrintData

`PrintData()`

Corresponds in the interface to:

- Right mouse button in column heading of list, Print: Print.

PrintPreview

```
PrintPreview()
```

Corresponds in the interface to:

- Right mouse button in column heading of list, Print: Page preview.

PageSetup

```
PageSetup()
```

Corresponds in the interface to:

- Right mouse button in column heading of list, Print: Page setup.

AppendRowObject

```
AppendRowObject(ByVal MainRowObject as Object)
```

- MainRowObject: The row object.

A row object is appended to the QueryBrowser list. The new row is also added to the OrigCollection (Page 17).

DeleteRowObject

```
DeleteRowObject(ByVal MainRowObject as Object)
```

- MainRowObject: The row object.

A row is removed from the `QueryBrowser` and the OrigCollection (Page 17); the Comos object is not deleted.

RefreshRowObject

```
RefreshRowObject(ByVal MainRowObject as Object)
```

- `MainRowObject`: The row object.

Here you can refresh a row from which you do know the row object, but not the row index. The query will find the row for the specified row object and then refreshes the row.

EvalByValue

```
EvalByValue(Optional ByVal RowIndex as Long = -1)
```

- `RowIndex`: Row index.

Calls IQuery. EvalByValue (Page 20). See there for an explanation.

See also Default value "-1", Index: Start value (Page 12).

RefreshRow

```
RefreshRow(ByVal Index as Long, Optional ByVal RefreshData as
Boolean = True)
```

- `Index`: Row index.
- `RefreshData`: True: Forces a recalculation of the data.

Refreshes a row. See Refresh (Page 9): With RefreshData, you can force a recalculation of the data. RefreshRow is not available if the queries contain hierarchies (ISortItem.GroupLevel > 0).

See also Index: Start value (Page 12).

RefreshCell

```
RefreshCell(ByVal RowIndex as Long, ByVal ColIdent as Variant,
Optional ByVal RefreshData as Boolean = True)
```

- `RowIndex`: Row index.
- `ColIdent`: (Integer or String): Either Index from 1 to ColumnCount or the column name. Input difference: see Name (Page 42).
- `RefreshData`: True: Forces a recalculation of the data.

Not supported for hierarchies (...ISortItem.GroupLevel>0)

See *RefreshRow*.

See also Index: Start value (Page 12).

1.23.2 IQueryBrowser: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

Query

```
Query() as IQuery
```

Here, the `Query` is made known; there is no counterpart. The QueryBrowser therefore knows the `Query`, but the `Query` does not know the QueryBrowser.
See also IQuery (Page 20)

SelectedColumns

```
SelectedColumns() as Object Read only.
```

Collection of the selected columns.

See Collection (Page 12).

CurrentColumn

`CurrentColumn() as IColumnDef Read only.`

The current column. If multiple columns are selected, the `CurrentColumn` is the column where the selected cell is located.
See also IColumnDef (Page 41)

SelectedObjects

`SelectedObjects() as Object Read only.`

Results from the selection in the QueryBrowser.

See Collection (Page 12).

CurrentObject

`CurrentObject() as Object Read only.`

The current object. If multiple objects are selected, the `CurrentObject` is the object with a gray frame.

SelectedRowObjects

`SelectedRowObjects() as Object Read only.`

Collection of row objects of the selected rows. Results from the selection in the QueryBrowser: The query knows the selected rows and identifies the respective row objects.

See Collection (Page 12).

CurrentRowObject

`CurrentRowObject() as Object Read only.`

Row object of the row with the CurrentObject.

Function bundle CurrentQuery, CurrentQueryRowIndex, CurrentQueryColumnIndex.

CurrentQuery

`CurrentQuery() as IQuery Read only.`

Only has an effect for hierarchical nested object queries: The current query, based on the selection. Without hierarchy, `CurrentQuery` is always `Query`.

See also IQuery (Page 20)

CurrentQueryRowIndex

`CurrentQueryRowIndex() as Long Read only.`

Depending on the current selection, you find the `CurrentQuery` and the query there identifies the row involved in the `CurrentQuery` .

CurrentQueryColumnIndex

`CurrentQueryColumnIndex() as Integer Read only.`

Depending on the current selection, you find the `CurrentQuery` and the query there identifies the column involved in the `CurrentQuery`.

ActionTextShow

```
ActionTextShow() as Boolean
```

Shows or hides the script editor with the "Action" script block. Corresponds in the interface to the "Script block Action: Display" icon switch.

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

Here it is checked if the QueryBrowser was changes, thus the settings and so on. A simple change of the displayed data does not trigger an `IsChanged`.

RecordSelectors

```
RecordSelectors() as Boolean
```

Switches the dataset selectors on or off. Corresponds in the interface to: Options, "General" tab: Dataset selectors.

RowHeight

```
RowHeight() as Long
```

Determines the row height in complete steps. Corresponds in the interface to: Options, "General" tab: Row height. Always affects all rows.

UpdateWatch

```
UpdateWatch() as Boolean
```

Reacts to the COMOS Collection Workset.ChangedObjects. If UpdateWatch is enabled, the query is updated from the outside. Corresponds in the interface to: Options, "General" tab: Change monitoring.

The Reimport and Translate queries do not participate here; besides, CreateNewRow (Page 27) is not involved.

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

See also IStringStorage (Page 59)

Owner

```
Owner() as ITopQueryBrowser Read only
```

Owner (Page 11).

See also ITopQueryBrowser (Page 65)

ContainingQuery

`ContainingQuery(ByVal x as Long, ByVal y as Long) as IQuery Read only.`

Note

Here Query is determined, not the QueryBrowser. The QueryBrowser is already known at this point, since ContainingQuery is generated by an IQueryBrowser instance.

Several queries can be active in a QueryBrowser, e.g. hierarchical object queries. Therefore, both the X and Y coordinates are required to identify the current query.

See Containing (Page 72).

See also IQuery (Page 20)

ContainingQueryRowIndex

`ContainingQueryRowIndex(ByVal x as Long, ByVal y as Long) as Long Read only.`

Index of a row in the `ContainingQuery`. See Containing (Page 72).

See also Index: Start value (Page 12).

ContainingQueryColumnIndex

`ContainingQueryColumnIndex(ByVal x as Long, ByVal y as Long) as Integer Read only.`

Index of a column in the `ContainingQuery`. See Containing (Page 72).

See also Index: Start value (Page 12).

ContainingObject

`ContainingObject(ByVal x as Long, ByVal y as Long) as Object Read only.`

See Containing (Page 72).

Redundant; can also be determined by row and column

ContainingColumn

`ContainingColumn(ByVal x as Long) as IColumnDef Read only.`

QueryBrowser column. As Containing is always activated within a QueryBrowser, one coordinate is sufficient here to determine the column.

See Containing (Page 72).

See also IColumnDef (Page 41)

ContainingRowObject

`ContainingRowObject(ByVal y as Long) as Object Read only.`

QueryBrowser row object. As Containing is always activated in a QueryBrowser, one coordinate is enough here to determine the row object.

See Containing (Page 72).

WorkingType

`WorkingType() as qeWorkingType`

Corresponds in the interface to: Options, General tab, "Cell-oriented" or "Row-oriented".

See also qeWorkingType (Page 114)

MultiSelect

`MultiSelect() as Boolean`

Corresponds in the interface to: Options, General tab, "multi-selection".

ProfileMaster

`ProfileMaster() as IProfileMaster`

The profile technology is used to save settings. You can find more information on this topic in the "COMOS Administration" manual, keyword "Base objects "User settings" (Profiles)".

The profile technology is not a part of the query technology; it is only used by query. The profile technology is used at various places in COMOS.

See also section Interfaces of the profile technology (Page 101).

See also IProfileMaster (Page 101).

DragOverSelectedObjects

`DragOverSelectedObjects() as Object Read only.`

If you have a selection, then the input via coordinates is no longer allowed for a drag&drop operation. All cells that don not belong to the selection are blocked (even if these could be set without selection). Instead, you may only set objects in the selection and the coordinates are evaluated only there.

Firstly, it will be attempted to apply the drag&drop operation to all objects in the selection, and only after that the entry check will follow and disallowed entries will be rejected.

See Collection (Page 12).

DragOverSelectedRowObjects

`DragOverSelectedRowObjects() as Object Read only.`

Applies to the case if row objects are selected. Otherwise, see `DragOverSelectedObjects` above.

MappingInfoShow

`MappingInfoShow() as Boolean`

Practical to use only in connection with XML connectors. Enables an additional column heading in the browser, which is inserted above the normal column titles. It displays the mapping of the query (i.e. of the XML connector) to an XMLPath.

HeadHeight

`HeadHeight() as Long`

Height of column headings. Corresponds in the interface to: Options, "General" tab, "Headline size". Does not affect the height of MappingInfoShow.

EvalByValueType

`EvalByValueType() as qeEvalByValueType`

See Enums. Corresponds in the interface to: Options, "General" tab, "Apply immediately".

See also qeEvalByValueType (Page 114)

FixedColumns

`FixedColumns() as Integer`

Corresponds in the interface to: Options, "General" tab, "Number of fixed columns".

SelectedRowIndexs

```
SelectedRowIndexs(Optional ByVal HIndex as Integer = -1) as Object
Read only.
```

- `HIndex`: Index of the hierarchy level
  See alsoDefault value "-1", Index: Start value. (Page 12)

Returns the indices of the selected rows. In a hierarchical query, a selection in the "inside" part automatically also refers to rows in the higher hierarchical levels, so that you must specify which hierarchy level is implied.

See Collection (Page 12).

SelectedSubQueries

```
SelectedSubQueries(Optional ByVal HIndex as Integer = -1) as Object
Read only.
```

- `HIndex`: Index of the hierarchy level
  See alsoDefault value "-1", Index: Start value. (Page 12)

Here, depending on the hierarchy level, you can get the relevant query. Redundant: Among other things, you could also get the `ContainingQuery` in this way.

See Collection (Page 12).

HInitState

`HInitState() as qeHState`

See Enums. Corresponds in the interface to: Options, "General" tab, "Hierarchy levels unfolded". The initial state cannot be mixed. However, while working, mixed states are allowed as well.

See also qeHState (Page 114)

HWorkState

```
HWorkState() as qeHState
```

Current state of HState, may also be mixed.

See also qeHState (Page 114)

PreEval

```
PreEval() As Boolean
```

Corresponds in the interface to: Options, "General" tab: "Background loading" option

Enables or disables "Background loading for cells". Evaluates the `BackgroundLoadPercentageLimitLoadedObjects` option.

PreEvalLimit

```
PreEvalLimit() As Integer
```

Corresponds in the interface to: Options, "General" tab: "Limit" option

Entry for the limit as a percentage value of "Maximum no. of loaded objects". The value can only be between 1 and 100. Background loading stops when the limit is reached.

1.24 IQBrowserEvents

1.24.1 IQBrowserEvents: Sub

Change

```
Change()
```

Is triggered when the selection in the browser is changed.

BeforeColumnsDelete

```
BeforeColumnsDelete(ByVal Columns as Object, ByRef Cancel as
Boolean)
```

- `Columns`: VB collection of type IColumnDefs (Page 39).
- `Cancel`: Cancels the deletion.

Is triggered when columns are to be deleted. In the user interface, this is only possible with the right mouse button: "Delete".

AfterColumnsDelete

```
AfterColumnsDelete()
```

Event after deleting the columns.

BeforeRowsDelete

```
BeforeRowsDelete(ByVal RowObjects as Object, ByRef Cancel as
Boolean)
```

- `RowObjects`: VB collection of type `RowObjects`
- `Cancel`: Cancels the deletion.

The deletion of rows is normally not available in the user interface: direct deletion of rows is not available in the standard queries.

Internally, `BeforeRowsDelete` is used in terms of `ITopQueryBrowser` to `IQueryBrowser`: `ITopQueryBrowser` is an Event handler of `IQueryBrowser` (in everyday language: the QueryBrowser tells the TopQuery what has changed).

AfterRowsDelete

```
AfterRowsDelete(ByVal RowObjects as Object)
```

- `RowObjects`: VB collection of type `RowObjects`

Event after deleting the rows.

OLECompleteDrag

```
OLECompleteDrag(ByRef Effect as Long)
```

See *OLEDragDrop*.

OLEDragDrop

```
OLEDragDrop(ByVal Data as Object, ByRef Effect as Long, ByRef Button
as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y as Long)
```

Here the OLE technology of Microsoft is applied, the data involves a Microsoft OLE object (and not a Comos object).
For this, see MSDN Help, keyword: "OLEDragDrop" or "OLE DragDrop".

OLEDragOver

```
OLEDragOver(ByVal Data as Object, ByRef Effect as Long, ByRef Button
as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y as Long,
ByRef State as Integer)
```

See *OLEDragDrop*.

OLEStartDrag

```
OLEStartDrag(ByVal Data as Object, ByRef AllowedEffects as Long)
```

See *OLEDragDrop*.

OLETitleDragDrop

```
OLETitleDragDrop(ByVal Data as Object, ByRef Effect as Long, ByRef
Button as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y
as Long)
```

See *OLEDragDrop*.

OLETitleDragOver

```
OLETitleDragOver(ByVal Data as Object, ByRef Effect as Long, ByRef
Button as Integer, ByRef Shift as Integer, ByRef x as Long, ByRef y
as Long, ByRef State as Integer)
```

See *OLEDragDrop*.

DblClick

```
DblClick(ByRef IsValid as Boolean)
```

- `IsValid`: Specifies whether double-clicking is allowed.

Ultimately, the "Double-click" operating system event is offered here.

PopUp

```
PopUp(ByVal PopUpMenu as Object, ByVal DeadArea as Boolean)
```

- `PopUpMenu`: Object of the popup.ocx class.
- `DeadArea`: The DeadArea is the "white area" where no columns are displayed. For example, if only two columns are displayed in the initial state of the object query, the entire dialog window has another white area on the right where no table is displayed. A separate mouse menu can be generated for the DeadArea.

You can find additional information on this topic in the "COMOS Platform Administration" manual, keyword "OnMenuCreate (Popup, Context)".

PopUpSubMenu

```
PopUpSubMenu(ByVal PopUpMenu as Object, ByVal ID as String)
```

This event is not responsible for all submenus, but only for the submenus that are generated a little later, for example because some objects still need to be calculated. Example: Navigation menu.

In PopUp, submenus can also be generated.

You can find additional information on this topic in the "COMOS Platform Administration" manual, keyword "OnSubMenuCreate (Popup, ID, Context)".

PopUpCmd

```
PopUpCmd(ByVal PopUpMenu as Object, ByVal ID as String)
```

- `PopUpMenu`: Object of the popup.ocx class.
- `ID`: Unique key of a menu command, set by the developer.

These are individual commands of the popup menu. The event is triggered when you click a command.

TitlePopUp

`TitlePopUp(ByVal PopUpMenu as Object, ByVal ColumnDefI as IColumnDef)`

- `PopUpMenu`: Object of the popup.ocx class.
- `ColumnDefI`: Column in whose heading the mouse menu is activated.

PopUp menu for the column headings. For application, see *PopUp*.

See also IColumnDef (Page 41)

TitlePopUpSubMenu

`TitlePopUpSubMenu(ByVal PopUpMenu as Object, ByVal ID as String, ByVal ColumnDefI as IColumnDef)`

- `PopUpMenu`: Object of the popup.ocx class.
- `ID`: Unique key of a submenu, set by the developer.
- `ColumnDefI`: Column in whose heading the mouse menu is activated.

Counterpart of *PopUpSubMenu*, but here for the mouse menu in the column header.

See also IColumnDef (Page 41)

TitlePopUpCmd

`TitlePopUpCmd(ByVal PopUpMenu as Object, ByVal ID as String, ByVal ColumnDefI as IColumnDef)`

- `PopUpMenu`: Object of the popup.ocx class.
- `ID`: Unique key of a submenu, set by the developer.
- `ColumnDefI`: Column in whose heading the mouse menu is activated.

Counterpart of *PopUpCmd*, but here for the mouse menu in the column header.

See also IColumnDef (Page 41)

AfterCellEdit

`AfterCellEdit(ByVal ColumnDef as IColumnDef, ByVal CurrentQuery as IQuery, ByVal RowIndex as Long, ByVal ColumnIndex as Integer)`

- `ColumnDef`: Current column
- `CurrentQuery`: Current Query
- `RowIndex`: Row index
- `ColumnIndex`: Column index

Event upon leaving the cell.

See also `Index: Start value`.

See also IColumnDef (Page 41)

See also IQuery (Page 20)

BeforeCellEdit

```
BeforeCellEdit(ByVal ColumnDef as IColumnDef, ByVal CurrentQuery as
IQuery, ByVal RowIndex as Long, ByVal ColumnIndex as Integer, ByRef
Cancel as Boolean)
```

- `ColumnDef`: Current column
- `CurrentQuery`: Current Query
- `RowIndex`: Row index
- `ColumnIndex`: Column index
- `Cancel`: Cancels the edit process, the cell remains unchanged.

This triggers the edit mode; editing does not have to be started yet.

See also `Index: Start value`.

See also IColumnDef (Page 41)

See also IQuery (Page 20)

BeforeCreateStdPopUp

```
BeforeCreateStdPopUp(ByVal PopUpMenu as Object, ByVal DeadArea as
Boolean, ByVal ID as String, ByRef Cancel as Boolean, ByRef
ComosObjects as Object, ByRef Disabled as Boolean)
```

- `PopUpMenu`: Object of the popup.ocx class.
- `DeadArea`: The DeadArea is the "white area" where no columns are displayed. For example, if only two columns are displayed in the initial state of the object query, the entire dialog window has another white area on the right where no table is displayed. A separate mouse menu can be generated for the DeadArea.
- `ID`: Unique key of a menu command, set by the developer.
- `Cancel`: The menu command is created (cancel = false) or not created (cancel = true) in the popup menu.
- `ComosObjects`: Context in which the menu is activated
- `Disabled`: Disables the menu command.

The individual entries in the menu can be disabled depending on the context (i.e. made invisible).

HeadClick

```
HeadClick(ByVal ColumnDef as IColumnDef)
```

- `ColumnDef`: Column in whose heading a selection is made.

Event upon selecting a column heading.

AfterRowsMove

`AfterRowsMove(ByVal FirstIndex as Long, ByVal MovedRowCount as Long)`

- `FirstIndex`: Index of the first row to be moved
- `MovedRowCount`: Number of to be moved rows

Only successive rows can be moved. In the user interface of the standard queries, it is not possible to move rows.

See also `Index: Start value.`

ColumnsChanged

`ColumnsChanged()`

Event on changing a column. Pertains to changes in the column properties, i.e. width, position etc.

1.24.2 IQBrowserEvents: Property

Version

`Version() as Integer Read only`

Version (Page 10).

1.25 IPartners

Also see Partners (Page 67).

Partners are other Comos components that can communicate with `TopQueryBrowser`. At present, these are:

- Bulk processing
- Product data selection

The partners can supply their setting via IStringStorage.

1.25.1 IPartners: Function

Add

```
Add(ByVal Partner as Object, ByVal ID as String, ByVal Description
as String) as IPartner
```

Logs a partner into the Collection.

See also IPartner (Page 87)

Remove

```
Remove(ByVal ItemIdent as Variant) as Boolean
```

Removes a partner from the Collection.

Count

```
Count() as Integer
```

Count of the Collection.

Item

```
Item(ByVal ItemIdent as Variant) as IPartner
```

An element in the Collection.

See also IPartner (Page 87)

1.25.2 IPartners: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

1.26 IPartner

1.26.1 IPartner: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Dispatch

```
Dispatch() as Object Read only
```

Dispatch (Page 10).

ID

`ID() as String Read only.`

Unique key; best a ProgID. The ID is determined by the developer during the implementation; thereafter it is Read only.

Description

Description() as String Read only.

Free description. Example in the interface: Bulk processing, Options: Administration tab, "Partner object" row.

PObject

`PObject() as Object Read only.`

This is the partner object.

1.27 ITQBItems

Corresponds in the interface to: Options, Input layout tab.

1.27.1 ITQBItems: Sub

ShutDown

`ShutDown()`

ShutDown (Page 10).

1.27.2 ITQBItems: Function

Item

`Item(ByVal ItemIdent as Variant) as ITQBItem`

- `ItemIdent`: An element of the TopQueryBrowser controls.

See also ITQBItem (Page 89)

Copy

`Copy() as ITQBItems`

Copy (Page 10).

Use of Copy is not practical here and is available only for system reasons and for internal purposes.

1.27.3 ITQBItems: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Count

```
Count() as Integer Read only
```

Count of the Collection.

Owner

```
Owner() as ITopQueryBrowser Read only
```

Owner (Page 11).

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

In the interface, the only permitted change is to make a control visible.

1.28 ITQBItem

1.28.1 ITQBItem: Function

Copy

```
Copy() as ITQBItem
```

Copy (Page 10).

Use of Copy is not practical here and is available only for system reasons and for internal purposes.

1.28.2 ITQBItem: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Name

```
Name() as String Read only
```

Unique name; corresponds in the interface to: Options, Input layout tab, "Name" column.

Description

```
Description() as String Read only.
```

Corresponds in the interface to: Options, Input layout tab, "Description" column.

ItemType

```
ItemType() as qeTQBItemType Read only.
```

See Enum. Corresponds in the interface to: Options, Input layout tab, "Type" column.

See also qeTQBItemType (Page 115)

Picture

```
Picture() as Picture Read only.
```

Of type OLEPicture. Corresponds in the interface to: Options, Input layout tab, "Symbol" column or the displayed symbol in the toolbar.

Visible

```
Visible() as Boolean
```

Corresponds in the interface to: Options, Input layout tab, "Visible" column.

Owner

```
Owner() as ITQBItems Read only
```

Owner (Page 11).

IsChanged

```
IsChanged() as Boolean
```

IsChanged (Page 10).

In the interface, the only permitted change is to make a control visible.

1.29 IReImportAdmin

Only for internal purposes. Is a special feature of the "Reimport" query: Passes Office data.

1.29.1 IReImportAdmin: Function

TableList

```
TableList(ByVal FileName as String) as Object
```

Only for internal purposes.

DOMDocument

```
DOMDocument(ByVal FileName as String, ByVal TableName as String) as
IXMLDOMDocument
```

Only for internal purposes.

1.29.2 IReImportAdmin: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

1.30 IQCondition

1.30.1 IQCondition Function

- `Item(Index As Long) As IQCondition`
  States IQCondition.
- `AddNew([Index As Long = -1]) As IQCondition`
  Creates new IQCondition.
- `Delete(ItemIdent As Variant) As Boolean`
  Deletes IQCondition.
- `Remove(ItemIdent As Variant) As IQCondition`
  Deletes IQCondition from the list.

1.30.2 IQCondition Property

- `Version As Integer`
  States the version of the interface.
- `Attribute As String`
  See "IComosDSearchCondition.attribute".

- `AttributeType As String`
  See "IComosDSearchCondition.AttributeType".
- `ComparisonOperator As String`
  See "IComosDSearchCondition.ComparisonOperator".
- `LogicalOperator As String`
  See "IComosDSearchCondition.LogicalOperator".
- `PerformanceIndex As Long`
  See "IComosDSearchCondition.PerformancedIndex".
- `Value As String`
  See "IComosDSearchCondition.value".
- `Description As String`
  Description of the Condition.
- `InputByUser As qeInputByUser`
  Defines if the user has to enter something in the query or not, or if the user has to make an entry.
- `StdTableName As String`
  Regulates if the value came from a standard table.
- `ListValue As Boolean`
  ListValue = "True" means that values are taken over from a standard table.
- `StandardTable as IComosDStandardTable`
  Standard table can be assigned as an object.
- `SettingsValue(SetType As qeSettingsType, PropertyName As String)`
  Allows that a user value can be entered besides the standard value for the search which is defined by the administrator.
- `Count As Long`

1.30.3 IQCondition Sub

ResetSettings()

Sets the user value back to the standard value.

Sub Refresh()

Refreshes the object.

Clear()

`ResetSettings()`

Sub Refresh()

Interfaces for XML connectors 2

Additional information

- See alsoMapping (Page 28).
- See alsoIsKeyColumn (Page 42).
- See alsoMapping (Page 42).
- See alsoMappingInfoShow (Page 75)

2.1 IAdapterActions

2.1.1 IAdapterActions: Sub

PreAction

```
PreAction(jobType as Object, document as IComosDDocument, filename
as String, jobOptions as IOptions)
```

PostAction

```
PostAction(jobType as Object, document as IComosDDocument, filename
as String, jobOptions as IOptions)
```

2.2 IProgressBar

2.2.1 IProgressBar: Function

GetPercentage

```
GetPercentage(level As Long) As Double
```

Call the percentage number of a progress bar placed on the control. Both progress bars can be called separately via the ProgressBar. Allowed values are: 1 , 2.

2.2.2 IProgressBar: Property

State

```
State() As qeProgressState
```

Defines the status of a progress bar, is also used to display the status of the job.

Text

```
Text() As String
```

Version

```
Version() As Integer
```

Current version of the implemented interface.

2.2.3 IProgressBar: Sub

SetPercentage

```
SetPercentage(level As Long, Value As Double)
```

Setting the percentage number of a progress bar.

2.3 IMapping

Can only be used with XML connectors. Background: Can be used, for example, to map Comos units with a foreign spelling to a unit.

2.3.1 IMapping: Function

ValueByMapping

```
ValueByMapping(ByVal CellObject as Object, ByVal vNewValue as
Variant, ByVal CalcType as qeMappingCalcType) as Variant
```

- `CellObject`: Cell object. The cell object must already have been evaluated at this point.
- `vNewValue`: Simultaneously input and output, where `CalcTyp` specifies the direction in which the conversion takes place.
- `CalcType:` See Enum.

See also qeMappingCalcType (Page 115)

2.3.2 IMapping: Property

Version

`Version() as Integer Read only`

Version (Page 10).

MappingInfo

`MappingInfo() as String`

Depends on IBaseQuery (Page 26) and IColumnDef (Page 41).

ColumnDef.Mapping.MappingInfo: corresponds to the "XML-Mapping" field at the connector.

IBaseQuery. MappingInfo (Page 28): corresponds to the mapping at the connector in the "XML-Collection" field.

DeleteObjects

`DeleteObjects() as Boolean`

For future use.

In the XML connector, you can specify whether to delete objects whose counterpart no longer exists in the XML file. This setting applies for the entire connector.

DeleteObjects is a preparation to ensure that the delete instruction does not refer to the entire connector, but only to individual queries in the connector.

Table

`Table() as String`

SystemFullName for the standard table. The standard table must already exist at this point and must have been filled correctly. In this table, both the Comos units (Comos values) and the external values are entered.

ComosValueID

`ComosValueID() as Variant`

Function pair: *ComosValueID* and *ExternValueID*.

- `ComosValueID`: Column in the standard table in which the COMOS values are located.
- `ExternValueID`: Column in the standard talbe in which the external values are located.

The columns are identified as follows:

- String "Name" accesses the "Name" column (this is the far left column).
- String "Description" accesses the "Description" column (this is the second column).
- `XValueIndex`, thus 0 to 15 (in the user interface, "Column 1" is addressed with the index 0).

ExternValueID

```
ExternValueID() as Variant
```

ExternValueID must always be used together with *ComosValueID*. See there.

XmlTokenizedType

```
XmlTokenizedType() as qeXmlTokenizedType
```

Comos data type that corresponds to the XML data type. Currently supported data types: See Enum.

See also qeXmlTokenizedType (Page 115)

OrigCollectionType

```
OrigCollectionType() as qeOrigCollectionType
```

You can find more information on this topic in the "Interfaces" manual, keyword "Define COMOS Collection: REFID".

See also qeOrigCollectionType (Page 116)

IsCheckColumn

```
IsCheckColumn() as Boolean
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Check column".

2.4 IXMLConnectorJob

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

2.4.1 IXMLConnectorJob: Sub

Export

```
Export()
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

Import

```
Import()
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

2.4.2 IXMLConnectorJob: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

ComosObjects

```
ComosObjects() as Object
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

QueryContainer

```
QueryContainer() as IComosBaseObject
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

RootComosObject

```
RootComosObject() as IComosBaseObject
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

RootXMLNode

```
RootXMLNode() as IXMLDOMNode
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

TopQuery

```
TopQuery() as ITopQuery
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

See also ITopQuery (Page 17)

XMLNodes

```
XMLNodes() as IXMLDOMNodeList
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

RootXMLNodeCompare

```
RootXMLNodeCompare() as IXMLDOMNode
```

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

XMLNodesCompare

`XMLNodesCompare() as IXMLDOMNodeList`

You can find more information on this topic in the "COMOS Platform Interfaces" manual, keyword "Script tab".

2.5 IOption

2.5.1 IOption: Property

Version

`Version() as Integer Read only`

Version (Page 10).

Name

`Name() as String Read only.`

Name of the value, for example

Options.Item("ShowFile").Value = True

Value

`Value() as Variant`

The value that was passed.

Comment

`Comment() as String Read only.`

Can be used as script comment, for example.

2.6 IOptions

2.6.1 IOptions: Function

Item

```
Item(ByVal ItemIdent as Variant) as Ioption
```

See also IOption (Page 98)

2.6.2 IOptions: Property

Version

```
Version() as Integer Read only
```

Version (Page 10).

Count

```
Count() as Integer Read only.
```

Count of the Collection.

Interfaces of the profile technology 3

The profile technology is used to save settings. You can find more information on this topic in the "COMOS Platform Administration" manual, keyword "Administrating personal user settings (profile)".

The profile technology is not a part of the query technology; it is only used by query. The profile technology is used at various places in COMOS. In other words, the ivbQuery.dll not only contains query-specific interfaces, but also generally used interfaces.

A `ProfileMaster` component is the component for managing settings. It implements the `IProfileMaster` interface.

If want to use `ProfileMaster` as a developer, `IProfileStorage` must be implemented.

3.1 IProfileMaster

3.1.1 IProfileMaster: Sub

Save

```
Save()
```

Saves the profile object and thus also the XObj and the CDevice. The Comos object is found via *Name*.

Load

```
Load()
```

Comos object is found via *Name*.

3.1.2 IProfileMaster: Property

Version

```
Version() as Integer Read only
```

Version (Page 10)

Name

```
Name() as String
```

Name or `RelativName` of the base object; always related to a profile object as root. Please note: if you use `RelativName` in connection with creating a profile object, then `RelativName` can also generate a path.
The object is searched for in the following sequence:

- Engineering project: User
- Engineering project: All users
- Base project: User
- Base project: All users
- System project: User
- System project: All users

Description

`Description() as String`

Description of the Comos object at which the profile object is saved.
If multiple objects are generated using `RelativName`, then it is also possible to enter the descriptions of several objects, separating them with the pipe symbol.

ProgID

`ProgID() as String`

`ProgID` of the component that creates the profile. The developer must pass the ProdID himself. The ProgID then also determines which interface this profile object is allowed load.

Childs

`Childs() as Object`

See Collection (Page 12).
In addition to `Item` and `Count`, `Add` is allowed as well.
Here you can specify Childs for which profile objects are to be saved. Childs that are logged in here must have IProfileStorage implemented. If, for example, Save is triggered in the ProfileMaster, then the the IProfileStorage save is triggered in the Childs as well.

SType

`SType() as Long`

See Enums qeProfileType (Page 118).

Also see *STypeAllowed*.

Container

`Container() as IComosBaseObject Read only.`

This is the CDevice of the profile object. For permitted operations at this object see the comos_dll.pdf class documentation.

WorkSet

`WorkSet() as IComosDWorkset`

Comos Workset. For permitted operations at this object see the comos_dll.pdf class documentation.

StorageValue

`StorageValue(ByVal vName as String) as Variant Read only.`

- `vName`: Unique key for this value in the XML string of the XObj.

STypeAllowed

`STypeAllowed() as Long`

Here you determine which combinations are allowed for *SType*. To do this, you determine in which project saving is permitted, add the Enums thus permitted and pass the sum here. Any feasible combination of permitted Enums produces a unique sum.

3.2 IProfileStorage

3.2.1 IProfileStorage: Sub

Save

`Save(ByVal CurrentNode as IXMLDOMNode)`

An `IXMLDOMNode` is returned in which you can save as desired. An external component is used: `msxml.dll.domnode`

For this you need a project reference in VB to: `XMLStringStorage.dll`.If you have this project reference, you can write for example:

```
XML_WriteItem CurrentNode, "Text1", Text1.Text ' NO ITX
XML_WriteItem CurrentNode, "Text2", Text2.Text ' NO ITX
```

Load

`Load(ByVal CurrentNode as IXMLDOMNode)`

An `IXMLDOMNode` is returned in which you can save as desired. An external component is used: `msxml.dll.domnode`

For this you need a project reference in VB to: `XMLStringStorage.dll`. If you have this project reference, you can write for example:

```
Text1.Text = XML_ReadItem(CurrentNode, "Text1") ' NO ITX
Text2.Text = XML_ReadItem(CurrentNode, "Text2") ' NO ITX
```

3.2.2 IProfileStorage: Property

Version

`Version() as Integer Read only`

Version (Page 10).

Name

`Name() as String Read only.`

A unique name is set here in the course of the implementation, afterwards the archive name is of course `Read only`.

ArchiveVersion

`ArchiveVersion() as Integer Read only.`

Version of the implemented archive. The developer must supply this information himself.

The `ArchiveVersion` has nothing to do with Version of the interface.

4 Interfaces for import objects

The interfaces described in this section have no connection with the query technology. In other words, the `ivbQuery.dll` not only contains query-specific interfaces, but also generally used interfaces.

The following interfaces define the objects:

- New standard import: Table
- New standard import: XML

4.1 IImportObject

4.1.1 IImportObject: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10)

Init

```
Init()
```

Object is being initialized. Corresponds to opening the interface.

Execute

```
Execute()
```

Object is being executed. Corresponds to the Execute runtime mode.

4.1.2 IImportObject: Function

Copy

```
Copy() as IImportObject
```

Copy (Page 10)

4.1.3 IImportObject: Property

Version

`Version() as Integer Read only`

Version (Page 10)

ActiveConnection

`ActiveConnection() as String`

Set database file. In the user interface, corresponds to the "Database" field. Also see *Source*.

Source

`Source() as String`

Belongs to *ActiveConnection*. In the user interface, corresponds to the "Table / Query" field.

ScriptText

`ScriptText() as String`

This is the script block (bottom right in the user interface).

IsChanged

`IsChanged() as Boolean`

IsChanged (Page 10)

Is triggered when a setting or the script was changed. Any change in the imported data would not trigger an `IsChanged`.

Owner

`Owner() as Object Read only`

Owner (Page 11)
`CDevice.XObj.ImportObject` applies. Then the owner is the XObj.

WorkSet

`WorkSet() as IComosDWorkset`

Comos Workset. See *Kernel.pdf*.

Dispatch

`Dispatch() as Object Read only`

Dispatch (Page 10)

Locked

```
Locked() as Boolean
```

Locked (Page 11)
Does not exist like this in the interface. But if the "Key" is turned in the user interface, then this property is set.

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11)

ProtocolFileName

```
ProtocolFileName() as String
```

In the user interface, corresponds to the "Log file".

4.2 IImportBrowser

In the query technology there is a a calculation component (Query) and a display component (QueryBrowser).

The same is realized here: There is an internal object (ImportObject) and the user interface of the import tool (ImportBrowser). Even the classification is similar to the query technology: ImportBrowser knows ImportObject; but ImportObject does not know ImportBrowser.

4.2.1 IImportBrowser: Sub

ShutDown

```
ShutDown()
```

ShutDown (Page 10)

Refresh

```
Refresh()
```

Refresh (Page 9)

EventsHandlerAdd

```
EventsHandlerAdd(ByVal vNewValue as IImportBrowserEvents)
```

- `vNewValue`: The event to deregister.

EventsHandlerRemove

```
EventsHandlerRemove(ByVal vNewValue as IImportBrowserEvents)
```

- `vNewValue`: The event to deregister.

4.2.2 IImportBrowser: Property

Version

`Version() as Integer Read only`

Version (Page 10).

XObjContainer

`XObjContainer() as Object`

This is the CDevice.

IsChanged

`IsChanged() as Boolean`

IsChanged (Page 10).

ImportObject

`ImportObject() as IImportObject`

The ImportBrowser knows the ImportObject; but not vice-versa.

WorkSet

`WorkSet() as IComosDWorkset`

Comos Workset. See *Kernel.pdf*.

Dispatch

`Dispatch() as Object Read only`

Dispatch (Page 10).

ScriptEditor

`ScriptEditor() as Object Read only.`

For internal purposes. Provides the ScriptRTF.ocx.

Locked

`Locked() as Boolean`

Locked (Page 11).

In the user interface, corresponds to the "Key".

RunMode

```
RunMode() as ieImportRunMode
```

Without user interface, you are always in RunMode.

Storage

```
Storage() as IStringStorage Read only
```

Storage (Page 11).

4.3 IImportBrowserEvents

4.3.1 IImportBrowserEvents: Sub

Change

```
Change()
```

Is triggered if settings were changed in the ImportBrowser.

4.3.2 IImportBrowserEvents: Property

Version

```
Version() as Integer Read only
```

Version (Page 10)

5 ivbQuery constants

5.1 qeColumnDefProp

qcUndef	0
qcDescription	1
qcName	2
qcFullName	3
qcSystemFullName	4
qcNestedName	5
qcDescription2	6
qcDescription3	7
qcIsFolder	8
qcLock	9
qcLabel	10
qcFullLabel	11
qcRelativLabel	12
qcSignedLabel	13
qcAliasFullLabel	20
qcAliasRelativLabel	21
qcAliasSignedLabel	22
qcDisplayValue	30
qcGetDisplayXValue	31
qcValue	32
qcGetXValue	33
qcProductValue	34

5.2 qeColumnAlignment

qcColumnAlignmentLeft	0
qcColumnAlignmentRight	1
qcColumnAlignmentCenter	2

5.3 qeColumnSizeUnit

qcColumnUnitRowHeight	0
qcColumnUnitTwips	1
qcColumnUnitMM	2

5.4 qeConnectorJobType

qcImport	0
qcExport	1

5.5 qeDialogType

qcNoDialog	0
qcFullDialog	1
qcBarDialog	2

5.6 qeInheritMode

qcInheritModeOwner	0
qcInheritModeChild	1
qcInheritModeMix	2

5.7 qeInputByUser

qcNoInput	0
qcOptional	1
qcRequired	2

5.8 qePresentation

qcNormal	0
qcComboList	1
qcComboEdit	2

qcCheckBox	3
qcCheckDot	4
qcSelectButton	5

5.9 qeProgressState

qcStateClosed	0
qcStateExecuting	1
qcStateAbort	2

5.10 qeVItemsSortType

qcSTNone	0
qcSTNumeric	1
qcSTAlphabetic	2
qcSTAlphabeticCaseSensitive	3

5.11 qePermissions

qeNone	0
qeReadOnly	1
geReadWrite	2

5.12 qeIFilterItemType

qcExpression	0
qcOperatorAnd	1
qcOperatorOr	2
qcOperatorNot	3
qcBracketOpen	4
qcBracketClose	5

5.13 qelSortOrder

qcSortAsc	0
qcSortDesc	1
qcSortNot	2

5.14 qelSortType

qcAlphaNumeric	0
qcAlphabetic	1
qcNumeric	2

5.15 qeWorkingType

qcByCell	0
qcByRow	1

5.16 qeEvalByValueType

qcEvalByRequest	0
qcEvalByCell	1

5.17 qeHState

qcHClose	0
qcHOpen	1
qcHMixed	2

5.18 qeStyleType

qcStandardStyle	0
qcCellStyle	1

5.19 qeExportType

qcExportTypeTxt	0
qcExportTypeXls	1
qcExportTypeXml	2
qcExportTypeMdb	3

5.20 qePictureType

qcOriginalPicture	0
qcBMPPicture	1

5.21 qeTQBItemType

qcButton	0
qcComboBox	1

5.22 qeEditType

qcDirect	0
qcEvalByValue	1

5.23 qeMappingCalcType

qcComosToExtern	0
qcExternToComos	1

5.24 qeXmlTokenizedType

qcID	0
qcIDREF	1
qcNone	12

5.25 qeOrigCollectionType

qcCollStandard	0
qcCollREFIDObjects	1

5.26 qeValueType

qcLong	3
qcDouble	5
qcDate	7
qcString	8
qcVariant	12
qcBoolean	11

5.27 qeRunObjectType

qcRObjUndef	0
qcRObjCellObject	1
qcRObjTopQueryMainObject	2
qcRObjProject	3
qcRObjSystemFullName	4
qcRObjObject	5

5.28 qeCalculateType

qcCalculateStandard	0
qcCalculateAfterFilterSort	1

5.29 ieImportRunMode

icDesign	0
icRunTime	1

5.30 QueryIVersion

qcIColumnDef_Version	2
qcBMPPic	1
qcIColumnEval_Version	2
qcIStringStorage_Version	2
qcIColumnDefs_Version	1
qcIFilterItem_Version	1
qcIFilter_Version	1
qcISortItem_Version	1
qcISort_Version	1
qcIBaseQuery_Version	1
qcIQuery_Version	2
qcIQueryBrowser_Version	1
qcICell_Version	1
qcITopQuery_Version	2
qcITopQueryBrowser_Version	1
qcIExtended_Version	10
qcIPartners_Version	1
qcIPartner_Version	1
qcIQBrowserEvents_Version	10
qcITopQBrowserEvents_Version	3
qcIValueItem_Version	1
qcIValueItems_Version	1
qcIStyle_Version	1
qcIImportObject_Version	1
qcIImportBrowser_Version	1
qcIImportBrowserEvents_Version	1
qcITQBItems_Version	1
qcITQBItem_Version	1
qcIReImportAdmin_Version	1
qcIProfileMaster_Version	1
qcIProfileStorage_Version	1
qcIRunObjectDef_Version	1
qcIOption_Version	1
qcIOptions_Version	1
qcIXMLConnectorJob_Version	1
qcIMapping_Version	2

5.31 qeColumnExtendedType

qcColumnExtendedNone	0
qcColumnExtendedObject	1
qcColumnExtendedValue	2
qcColumnExtendedPicture	4
qcColumnExtendedStyle	8
qcColumnExtendedValueItems	16
qcColumnExtendedObjectByValue	32

5.32 qeIFilterItemOperator

qcEQ	2
qcGT	3
qcGE	4
qcLT	5
qcLE	6
qcNE	7
qcInclude	9
qcVBLike	10

5.33 qeProfileType

qcGeneral	0
qcCurrentProject	1
qcCDeviceProject	2
qcCurrentUser	4
qcAllUsers	8
qcSystemProject	16

5.34 qeObjectByValueStdOptions

qcObjectByValueStandard	0
qcObjectByValueOwnerByCDev	1