SIEMENS

SIMATIC S7

Toolbox for PGs and PCs (AT and compatibles)

P R O D A V E   S 7   V 5 . 6

Toolbox for WINDOWS 95/98/NT/ME/2000/XP

Applications for a Data Link

of PGs/PCs to SIMATIC S7 via MPI Interface and

PC/MPI Cable

Operating Instructions

PRODAVE S7 Win95/98/NT/ME/2000/XP mini	**Order No.: 6ES7 807 - 3BA00 - 0YA0**
PRODAVE S7 Win95/98/NT/ME/2000/XP	**Order No.: 6ES7 807 - 4BA00 - 0YA0**

Index

1. Introduction

Due to their constantly increasing performance and vast availability of PC applications for the manufacturing process, the Personal Computer is being used more and more on the factory shop floor in addition to the programming unit. This, however, poses the problem to you as the user how to combine the variety of programs for handling of process data (e.g. data banks, statistical evaluation) with your existing PLC systems. In order to make PLC data available for the PC application you will need a working and cost effective data link between PLC and PC.

This is where the software package PRODAVE S7 will offer the solution. PRODAVE S7 offers tried and tested functions (tools) in a DLL (Dynamic Link Library) or LIB (Library) which you can combine for each of your applications. The combination of the tools is carried out in programming languages for Windows 95/98/ME and Windows NT/2000.

Via these combined functions the process data traffic between PLC and PG/PC is established by PRODAVE S7 using the MPI interface of the PLC. The data now available can be translated into a format suitable for PCs and can be processed by your own application or any standard application. This will enable you to create a data link between PLC and PG/PC without having detailed knowledge, and all your development activities can be concentrated on specific processing of your data.

PRODAVE S7 enables you to not only evaluate and monitor but to influence your process as well inasmuch that you can have several functions available to you to enable you to write data to the PLC from the PG/PC.

As an introduction to PRODAVE S7 and to enable you to familiarise yourself with it, we supply several demonstration programs as examples. These functions are fully operational and are available in source code (see Para. 5. Demonstration Programs).

PRODAVE S7 runs under Windows 95/98/NT/ME/2000/XP on PG 7xx and on Pcs which are compatible to Industrial Standard in conjunction with MPI interfaces (CP5511, CP5611) or PC/MPI cables.

The PRODAVE functions can be divided into 3 basic types:

1.1 Basic Functions

- initialise and de-initialise system (load_tool, unload_tool).
- activate connection (new_ss).

1.1.1 Functions for Data Transfer to S7 300/400

- read output bytes from PLC (a_field_read).
- write output bytes (a_field_write).
- read input bytes from PLC (e_field_read).
- read data bytes from a block DB (d_field_read).
- write data bytes to a block DB (d_field_write).
- read flag bytes from PLC (m_field_read).
- write to flag bytes in PLC (m_field_write).
- status test of a flag (mb_bittest).
- set and reset flag (mb_setbit, mb_resetbit).
- read timer words from PLC (t_field_read).
- read counter words from PLC (z_field_read).
- overwrite counter words in PLC (z_field_write).
- read mixed data (mix_read).
- write mixed data (mix_write).

1.1.2 Functions for Data Transfer to S7 200

- read output bytes from PLC (as200_a_field_read).
- write output bytes (as200_a_field_write).
- read input bytes from PLC (as200_e_field_read).
- read data bytes from variable memory (as200_vs_field_read).
- write data bytes to variable memory (as200_vs_field_write).
- read flag bytes from PLC (as200_m_field_read).
- write to flag bytes in PLC (as200_m_field_write).
- read special flag bytes from PLC (as200_sm_field_read).
- write to special flag bytes in PLC (as200_sm_field_write).
- status test of a flag (as200_mb_bittest).
- set and reset flag (as200_mb_setbit, as200_mb_resetbit).
- read timer words from PLC (as200_t_field_read).
- read counter words from PLC (as200_z_field_read).
- overwrite counter words in PLC (as200_z_field_write).
- read mixed data (as200_mix_read).
- write mixed data (as200_mix_write).

1.2 Functions for Data Handling in PG/PC

- error text output relating to error number (error_message).
- format conversion of S7 data (gp_to_float, float_to_gp).
- format conversion of S5 data (kg_to_float, float_to_kg).
- byte conversion of a byte to eight logical values and vice versa (boolean_byte, byte_boolean).

1.3 TeleService Functions

The TeleService functions are an expansion of the PRODAVE functionality which enables the user to establish a connection of and to an S7 controller via the public telephone network.
Pre-requisite is the installation of the SIMATIC TeleService =
SW-Option package to STEP 7 for the linking of SIMATIC S7 controllers (PLCs) via the public telephone network.

- Dial a station and / or a TS adaptor (ts_dial).

- Close a TeleService connection (ts_hang_up_dial).

- Initialise the system for call recognition (ts_set_ringindicator).

- Read informationen on alarm triggering station (ts_read_info).

- Close a TeleService connection (ts_hang_up_ring).

2. Description

2.1 Operating Mode of PRODAVE

Using the programming package PRODAVE S7 you can read data from a programmable logic controller (PLC) and write data to a PLC under Windows 95/98/NT/2000/XP via several CPUs from the S7-series.

PRODAVE S7 consists basically of two parts:

- driver for Windows 95/98/ME and Windows NT/2000/XP

and

- high language adaptor

PRODAVE S7 offers the adaptor for Windows 95/98/NT/ME/2000/XP in the form of a 32-Bit-DLL (Dynamic Link Library) created in VC++ Version 6.0.

If you wish to read data from the PLC or write data to the PLC using a high language, you will only require the adaptor and its functions.

2.2 Use of the High Language Adaptor

A detailed description of the available functions for the various programming languages of this manual can be found in Chapter 4 "Description of the PRODAVE Functions".

2.3 Pre-requisites

PRODAVE S7 operates with the following PLC types: S7-200, S7-300, S7-400, M7 and C7 from the S7 series.

Software-Prerequisites:

Operating System Windows 95/98/ME or Windows NT V4.x/2000/XP.

Hardware-Prerequisites:

PRODAVE S7
PRODAVE S7 mini

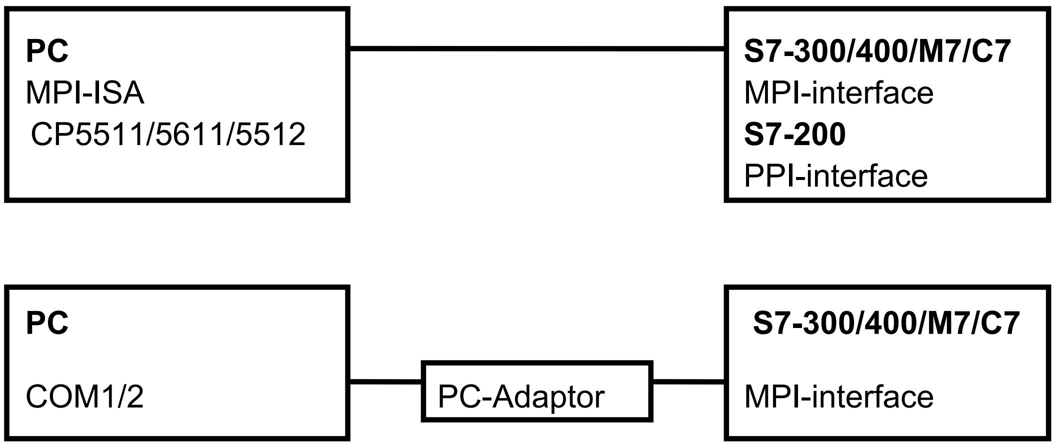
Simatic PG or AT compatible Industrial-PC with 64MB main memory and MPI-ISA-interface, CP5511, CP5611, CP 5512 or PC-Adaptor.

2.4 Connection of PG/PC to PLC

2.4.1 Driver under Windows 95/98/NT/2000/XP

The PG/PC may be connected to the PLC by means of the following components:

- o CP 5611 PCI-Card
- o CP 5511 / 5512 PCMCIA-Card
- o MPI-ISA-Card or MPI-ISA on Board (Simatic PG, PC RI45,25,FI25)
- o COM 1/2 via PC-Adaptor



Installation and set-up of the required hardware is carried out via the STEP7-Tool **PG/PC interface parameterisation**, which is available in the system after successful installation.

3. Operation

3.1 Installation of PRODAVE S7

3.1.1 Installation of PRODAVE S7 under Windows 95/98/NT/ME/2000/XP

The installation of PRODAVE S7 is carried out via a Windows installation program (SETUP.EXE), which must be activated by the file manager under Windows.
After starting SETUP.EXE a destination path is offered for the installation which may be changed by new input or via BROWSE.
After specifying the destination path the following installation components are offered:

- PRODAVE S7 for Windows 95/98/NT/ME/2000/XP
  PRODAVE DLL and demonstration program for Windows 95/98/NT/ME/2000/XP.
  STEP7 Driver for Windows 95/98/NT/ME/2000/XP
- Documentation, Online Help.

Setup automatically generates an icon at the control panel to set up the used interface under Windows 95/98/ME/NT/2000.

The drivers to be used can be loaded, parameterised and linked into the system by means of the STEP7 tool **PG/PC-interface parameterisation** (S7EPATSX.EXE). After correct installation the drivers are automatically activated every time Windows 95/98/NT/ME/2000/XP is started.

3.2 Scope of Supply PRODAVE S7

3.2.1 PRODAVE S7 for Windows 95/98/NT/ME/2000/XP

The following PRODAVE components are available after successful installation:

SIEMENS\PRODAVE_S7\EXAMPLES\INCLUDE\

```
W95_S7    .H    = header file for PRODAVE-DLL
KOMFORT   .H    = header file for enhanced -DLL
W95_S7    .DEF  = definition file for PRODAVE-DLL
KOMFORT   .DEF  = definition file for enhanced -DLL
```

SIEMENS\PRODAVE_S7\EXAMPLES\LIB\

```
W95_S7    .LIB  = import library for PRODAVE-DLL
KOMFORT   .LIB  = import library for enhanced -DLL
```

SIEMENS\PRODAVE_S7\EXAMPLES\SAMPLE_VC\

```
DEMO      .EXE  = demo program
DEMO      .C    = source code demo program
ICON1     .ICO  = 32 x 32 icon
DEMO      .RC   = resource code demo program
ERROR     .DAT  = file with German error texts
RESOURCE.H      = header file demo program
```

SIEMENS\PRODAVE_S7\EXAMPLES \SAMPLE_VB\

```
VBDEMO    .VBP=   visual basic projectfile
ERROR     .DAT  = file with German error texts
VBDEMO    .EXE  = visual basic demo program
VBDEMO    .BAS  = visual basic file
VBDEMO          .FRM = visual basic FRM-file
DBBUCH_FRM      .FRM
ERROR           .FRM
FLAG            .FRM
INFO            .FRM
LOAD            .FRM
READ_FRM        .FRM
STATUS          .FRM
TS_FRM          .FRM
TSINFO_FRM      .FRM
WRITE_FRM       .FRM
```

\WINDOWS\SYSTEM32

```
W95_S7    .DLL  = PRODAVE-DLL
KOMFORT   .DLL  = enhanced -DLL
```

3.2.2 PRODAVE S7 mini for Windows 95/98/NT/ME/2000/XP

The following PRODAVE components are available after successful installation:

SIEMENS\PRODAVE_S7_MINI\EXAMPLES\INCLUDE\

W95_S7M	.H	= header file for PRODAVE-DLL
KOMFORT	.H	= header file for enhanced -DLL
W95_S7M	.DEF	= definition file for PRODAVE-DLL
KOMFORT	.DEF	= definition file for enhanced -DLL

SIEMENS\PRODAVE_S7_MINI\EXAMPLES\LIB\

W95_S7M	.LIB	= import library for PRODAVE-DLL
KOMFORT	.LIB	= import library for enhanced -DLL

SIEMENS\PRODAVE_S7_MINI\EXAMPLES\SAMPLE_VC_MINI\

DEMO	.EXE	= demo program
DEMO	.C	= source code demo program
ICON1	.ICO	= 32 x 32 icon
DEMO	.RC	= resource code demo program
ERROR	.DAT	= file with German error texts
RESOURCE.H		= header file demo program

SIEMENS\PRODAVE_S7_MINI\EXAMPLES\SAMPLE_VB_MINI\

VBDEMO	.VBP=	visual basic projectfile
ERROR	.DAT	= file with German error texts
VBDEMO	.EXE	= visual basic demo program
VBDEMO	.BAS	= visual basic file
VBDEMO		.FRM = visual basic FRM-file
DBBUCH_FRM		.FRM
ERROR		.FRM
FLAG		.FRM
INFO		.FRM
LOAD		.FRM
READ_FRM		.FRM
STATUS		.FRM
TS_FRM		.FRM
TSINFO_FRM		.FRM
WRITE_FRM		.FRM

\WINDOWS\SYSTEM32

W95_S7M	.DLL	= PRODAVE-DLL
KOMFORT	.DLL	= enhanced -DLL

3.3 Working with PRODAVE

The user program is written in a high language and the function calls are used in the form listed in Para. 4 "Description of PRODAVE Functions".

3.3.1 Notes on S7-200

When creating a data link to S7-200 it is not allowed to have more than one connection parameterised in the **load_tool** function.

The connection is **initialised** by means of the **load_tool** function. This is followed by the user specific part, where you may **only** call the **as200_......** functions from the adaptor (see also Para. 4.1.2 “Basic Functions for Data Transfer S7-200 “). When you wish to end your program, it is required to **de-initialise** the connections by means of the **unload_tool** function.

3.3.2 Notes on AS300/400

The obligatory start of each user program is the **initialisation** of the connections by calling the function **load_tool**. This is followed by the user specific part, where you can call any amount of PRODAVE functions (with the exception of the as200_.... functions) from the adaptor. When you wish to end your program, it is required to **de-initialise** the connections by means of the **unload_tool** function.

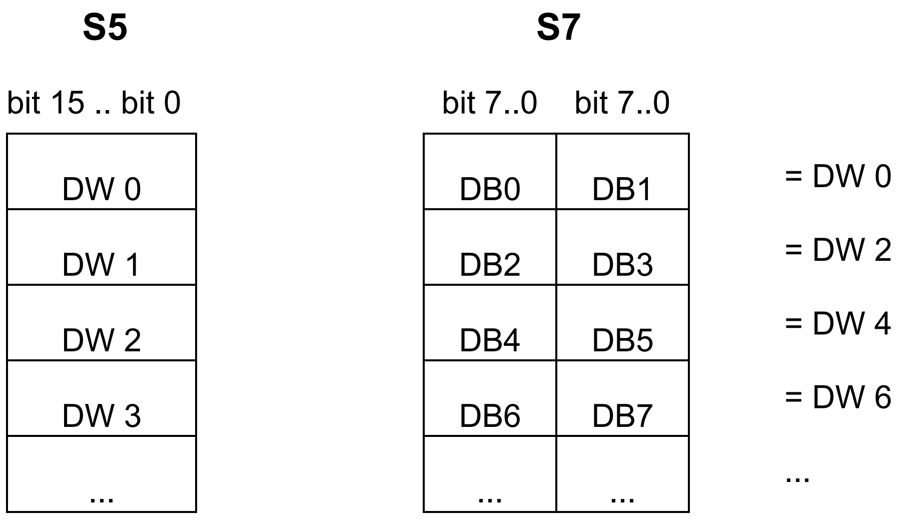
When carrying out the development of your program, the following points should be noted to avoid data loss or a system crash:

- Prior to leaving the program, the connections must be de-initialised by calling the adaptor function **unload_tool**!
- When reading data from the PLC, the fields into which data is to be transferred, must be big enough to receive this data as the adaptor does not carry out a field check!
- The error text file must be located in the same directory as the developed program as otherwise the adaptor will not be able to read the error texts!
- In order to avoid a repeated “check if it exists” of the error text file, you can call the function **error_message** at the start of the program to enable you to output an appropriate message in the event of an error. This error text file is loaded when calling this function for the first time.

- Prior to starting the PRODAVE application under MS-DOS it is required to activate the drivers for the used interface.

3.4 Differences between S5 and S7

The main difference between S5-PLCs and S7-PLCs is the management of data blocks. S5 data blocks are processed word by word, whereas the S7 data blocks are processed byte by byte.



When using the **d_field_read** function, the data block is accessed byte by byte such as, for instance, applies to the flag area.

When you read 3 data words using the **db_read** function, the PLC transfers DBW0 - DBW5. I.e. three 16bit words are available for processing in the PG/PC, which the PLC addresses via DBW0, DBW2 and DBW4, by the PG/PC, however, they are addressed via DW0, DW1 and DW2.

In order to avoid confusion in the data management, we recommend to have the PLC process the data block symbolically via type allocation in the following form:

Type Declaration in Symbol List:

Block: DB10
DB_10

Address	Variable	Data Type	Start Value	Comments
		STRUCT		
	DW	ARRAY[0 .. 255]		
		WORD		
		END STRUCT		

Example of access to the varilable in PLC:

```
L      “DB_10“.DW[2]
T      MW10
```

or

```
L      MW10
T      “DB_10“.DW[2]
```

3.5 Linking to Standard Tools

3.5.1 PRODAVE under Delphi (32-Bit) Example

To enable you to use the PRODAVE functions under Delphi, they must be declared as follows:

```
function Load_tool (  no:Byte;
                      name: {pointer} PChar;
                      adr:{pointer} PChar): longint;
stdcall;
external ´w95_s7.dll´ name ´load_tool´;

function DB_read (  dbno: longint;
                    dwno: longint;
                    var amount: longint;
                    var buffer): longint
stdcall;
external ´w95_s7.dll´ name ´db_read´;

function Unload_tool: longint;
stdcall;
external ´w95_s7.dll´ name ´unload_tool´;
```

Example:

```
var
    plc_adr_table : array [0..15] of byte;
    name:array[0..255] of char;
    res ,amount: longint;
    buffer : array[0..255] of word;

plc_adr_table[0] := 2;    {address}
plc_adr_table[1] := 0;    {segment id}
plc_adr_table[2] := 2;    {racktno}
plc_adr_table[3] := 0;    {slotno}
plc_adr_table[4] := 0;
strcopy(name,'S7ONLINE');
res := Load_tool(1,addr(name),addr(plc_adr_table[0]));
res := DB_read(10,0,amount,buffer);
res := Unload_tool;
```

3.5.2 PRODAVE under Access (32-Bit) Example

To enable you to use the PRODAVE functions under Access, they must be declared as follows:

```
Declare Function load_tool Lib “w95_s7.dll “ (     ByVal no As Byte,
                                                   ByVal name As String,
                                                   ByVal adr As String) As Long

Declare Function db_read Lib “w95_s7.dll “ (       ByVal dbno As Long,
                                                   ByVal dwno As Long,
                                                   ByRef amount As Long,
                                                   ByRef buffer As Integer) As Long

Declare Function unload_tool Lib “w95_s7.dll “ () As Long
```

Example:

```
Dim dbno As Long, dwno As Long, amount As Long
Dim buffer(50) As Integer
Dim plc_adr_table As String


res = load_tool 1,  “S7ONLINE“,  plc_adr_table,
res = db_read dbno, dwno, amount, buffer(0)
res = unload_tool
```

3.5.3 PRODAVE under Visual Basic (32-Bit) Example

To enable you to use the PRODAVE functions under Visual-Basic, they must be declared as follows:

```
Declare Function load_tool Lib "w95_s7.dll" (ByVal nr As Byte,
                                             ByVal dev As String,
                                             adr As plcadrtype) As Long

Declare Function db_read Lib “w95_s7.dll “ (    ByVal dbno As Long,
                                           ByVal dwno As Long,
                                           amount As Long,
                                           buffer%) As Long

Declare Function unload_tool Lib “w95_s7.dll “ () As Long
```

Example:

```
Dim dbno As Long, dwno As Long, amount As Long
Dim buffer(50) As Integer

Type plcadrtype
  adr As Byte
  SEGMENTID As Byte
  SLOTNO As Byte
  RACKNO As Byte
End Type

Dim plcadr (5) As plcadrtype

plcadr(0).adr = 2
plcadr(0).SEGMENTID = 0
plcadr(0).SLOTNO = 2
plcadr(0).RACKNO = 0
plcadr(1).adr = 0

res = load_tool (1, ”S7ONLINE”, plcadr)
res = db_read (dbno, dwno, amount, buffer(0))
res = unload_tool()
```

4. Description of PRODAVE Functions

How to read the descriptions of the functions:

- The explanantion of the functions is in "normal" type.

- This is usually followed by a note on functions where their handling is similar, or which are to be read in conjunction with the described function, in "**bold**" type and marked by ☞

- This is followed by the syntax for C functions in the "***`courier bold italics`*** " font.

- In the event of programming examples for the described adaptor function, these would again be printed in "***`courier bold italics`***" (C).

All functions in the 32-Bit-DLL W95_S7.DLL and W95_S7M.DLL have the suffix **WINAPI**. It is important to note this when using the functions as is it is not explicitly mentioned in the function description.

Functions included in the software package PRODAVE S7 mini have the suffix **mini** in the headline.

32-Bit-DLL for Windows 95/98/NT/ME/2000/XP

```
int     WINAPI a_field_read
        (int nr, int amount, void* buffer);
```

Functions Overview

Basic Functions	PRODAVE S7	PRODAVE S7 mini
load_tool	x	x
unload_tool	x	x
new_ss	x	x

Basic Functions S7-300/400	PRODAVE S7	PRODAVE S7 mini
ag_info	x	x
ag_zustand	x	
a_field_read	x	
a_field_write	x	
db_buch	x	
db_read	x	x
db_write	x	x
d_field_read	x	x
d_field_write	x	x
e_field_read	x	
m_field_read	x	
m_field_write	x	
mb_bittest	x	
mb_setbit	x	
mb_resetbit	x	
t_field_read	x	
z_field_read	x	
z_field_write	x	
mix_read	x	
mix_write	x	

Basic Functions S7-200	PRODAVE S7	PRODAVE S7 mini
As200_ag_info	x	x
as200_ag_zustand	x	
as200_a_field_read	x	
as200_a_field_write	x	
as200_vs_field_read	x	x
as200_vs_field_write	x	x
as200_sm_field_read	x	
as200_sm_field_write	x	
as200_e_field_read	x	
as200_m_field_read	x	
as200_m_field_write	x	
as200_mb_bittest	x	
as200_mb_setbit	x	
as200_mb_resetbit	x	
as200_t_field_read	x	
as200_z_field_read	x	
as200_z_field_write	x	
as200_mix_read	x	
as200_mix_write	x	

Enhanced Functions	PRODAVE S7	PRODAVE S7 mini
error_message	x	x
testbit	x	x
boolean_byte	x	x
byte_boolean	x	x
gp_to_float	x	x
float_to_gp	x	x
kg_to_float	x	x
float_to_kg	x	x
kf_integer	x	x

TeleService Functions	PRODAVE S7	PRODAVE S7 mini
ts_dial	x	
ts_hang_up_dial	x	
ts_set_ringindicator	x	
ts_read_info	x	
ts_hang_up_ring	x	

For the TeleService functions the prerequisite is the correct installation of software package SIMATIC TeleService V5.x.

4.1 Basic Functions

load_tool

This function initialises the adaptor, checks if the driver is loaded, initialises the parameterised addresses and switches the selected interface to active. Under under Windows 95/98/NT/ME/2000/XP it is possible to parameterise a maximum of 32 connections.

The following 3 parameters are transferred to this function:

no	⇨ number of active connection (1..32).
device	⇨ device name (zero terminated) of the used driver e.g. “S7ONLINE“ for the MPI driver or ZERO (default).
adr_table	⇨ pointer to address list of the connected party. In this instance “adr“ = 0 is interpreted as the end identifier of the list.

Structure Address List:

```
#pragma pack(1)
struct  {
        unsigned char  adr;           /* station address    default 2   */
        unsigned char  segmentid;     /* segment id          default 0   */
        unsigned char  slotno;        /* slot no             default 2   */
        unsigned char  rackno;        /* rack no             default 0   */
        } adr_table[5];
#pragma pack()
```

Each party is identified by means of an entry in the address list:

adr	station address of party
segmentid	segment ID of party = 0 (reserved for later expansions)
slotno	slot number of party
rackno	rack number of party = 0

A connected party is selected by means of the “no“ parameter of function **load_tool** and via the **new_ss** function:

connection 1 ⇒ new_ss(1) ⇒ adr_table[0];
connection 2 ⇒ new_ss(2) ⇒ adr_table[1];
connection 3 ⇒ new_ss(3) ⇒ adr_table[2];
: : :

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞See also unload_tool, new_ss

C-Adapter

```
int        load_tool
           (char nr,char* device,char* adr_table);
```

Example:
This example initialises 3 connections. Connection 1 to PLC with station address 2, connection 2 to PLC with station address 4 and connection 3 to PLC with station address 9. For any further program execution following “load_tool“ connection 3 is set to active.

C-Adapter

```
#include <w95_s7.h>
int   error;
adr_table_type plc_adr_table[3] = { {2,0,2,0},
                {4,0,2,0},
                {9,0,2,0},
                {0,0,2,0} };

:
:
res = load_tool(3,"S7ONLINE",plc_adr_table);
:
res = new_ss(1);     /* PLC No 1  address = 2 */
:
res = new_ss(2);     /* PLC No 2  address = 4 */
:
res = new_ss(3);     /* PLC No 3  address = 9 */
:
error = unload_tool();
```

new_ss

The function **new_ss** activates the connection of the PG/PC, which is to be used for the data exchange. The description of the connections and/or parties is transferred with the **load_tool** function.

no = 1 ⇨ connection 1 (connection address plc_adr_table[0])
no = 2 ⇨ connection 2 (connection address plc_adr_table[1])
no = 3 ⇨ connection 3 (connection address plc_adr_table[2])
no = 4 ⇨ connection 4 (connection address plc_adr_table[3])

Closed connections can be re-established with this function!

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞See also load_tool

C-Adapter ***(mini)***

```
int       new_ss (char no);
```

unload_tool

This function deinitialises the connections and the adaptor and must be re-called prior to leaving the application!

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞See also load_tool

C-Adapter ***(mini)***

```
int       unload_tool(void);
```

4.1.1 Basic Functions for Data Transfer S7-300/400

a_field_read

The function **a_field_read** reads "amount" of output bytes from the PLC starting from "no" and stores the read value into a variable field of the PG/PC.

If there were no errors, the function supplies 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also e_field_read, m_field_read**

C-Adapter

```
int        a_field_read (int no,int amount,void * buffer);
```

Example:
Output byte 10 is read. The read value is stored in "buffer".

C-Adapter

```
#include <w95_s7.h>

char   buffer;
int    error;
adr_table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };



error = load_tool(1,"S7ONLINE",plc_adr_table);
:
error = a_field_read(10,1,&buffer);
:
error = unload_tool();
:
:
```

a_field_write

The function **a_field_write** writes "amount" of bytes from the specified storage area into the PLC starting from output byte "no".

If there were no errors, the function supplies 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also m_field_write**

C-Adapter

```
int        a_field_write (int no,int amount,void* buffer);
```

ag_info

The function **ag_info** reads the issue level of the PLC software and the PG interface as well as the MLFB number of the PLC and stores them in a storage area of the PG/PC as an ASCII string zero-terminated. The issue levels must be interpreted as integer values, the MLFB numbers as ASCII values:

buffer[0] - buffer[1]: ⇨ Integer value ⇨ issue level PLC
buffer[2] - buffer[3]: ⇨ Integer value ⇨ issue level PGIF
buffer[4] - buffer[24]: ⇨ MLFB of connected PLC

If there were no errors, the function supplies 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also ag_zustand**

C-Adapter ***(mini)***

```
int       ag_info (void * buffer);
```

Example:
The issue levels of the PLC software, the PG interface as well as the PLC type are read.

C-Adapter

```
#include <w95_s7.h>

int   error;
adr_table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };
#pragma pack(1)
struct {
   unsigned short plcas
   unsigned short pgas;
   char  mlfb[21];
   } info;
#pragma pack()
:
error = load_tool(1,"S7ONLINE",plc_adr_table);
:
error = ag_info(&info);
:
error = unload_tool();
:
```

ag_zustand

This function reads the PLC status (RUN or STOP) from the PLC and stores the data in a storage area of the PG/PC.

buffer[0] = 0	⇨	AG is in RUN
buffer[0] = 1	⇨	AG is in STOP
buffer[0] = 1	⇨	AG is in RESTART

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞See also ag_info

C-Adapter

```
int        ag_zustand (void * buffer);
```

db_buch

The function **db_buch** checks which DBs exist in the PLC.
For this purpose a buffer of 512 words must be made available (see example).
If the value in buffer [123] is = 0 this means that block DB 123 does not exist in the PLC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int        db_buch (void * buffer);
```

Example:

The program checks if DB 123 exists in the PLC.

C-Adapter

```
#include <w95_s7.h>

unsigned short buffer[512];
int    error;

:
:
error = db_buch(buffer);
if (buffer[123] != 0)
   {
   /* DB 123 ist im AG vorhanden */
   }
:
error = unload_tool();
:
:
```

db_read

The function **db_read** reads an “amount“ of data words from a data block in the PLC and transfers them into a variable field of the PG/PC.
If the data block does not exist, this is indicated by a return value = error number.
If the data which is being read, exceeds the amount available in the data block, the " amount” is corrected and error message 303 hex is returned.

Important!
The data words are stored in the "buffer" not in accordance with Intel-Notation (low byte - high byte) but in STEP5-Notation (high byte - low byte). This is important if the data is processed further.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also db_write**

C-Adapter ***(mini)***

```
int       db_read
          (int dbno, int dwno, int* amount, void* buffer);
```

Example:
DB10 consisting of 100 data words (DW 0 - DW 99) exists in the PLC. 45 data words are to be read starting from DW5.

The read values are stored in the data buffer “buffer” and are available for processing.

C-Adapter

```
#include <w95_s7.h>

int   buffer[100];
int   error;
int   dbno,dwno,amount;

:
:
dbno   = 10;
dwno   = 5;
amount = 45;

error = db_read(dbno,dwno,&amount,buffer);
:
error = unload_tool();
:
```

db_write

The function **db_write** writes an amount of data words from a variable field of the PG/PC into the PLC.
If the data block does not exist this is indicated by a return value = error number.
If the data which is being written, exceeds the amount available in the data block, the " amount" is corrected and error message 303 hex is returned.

Important!
The data words must be stored in "buffer" in accordance with STEP5-Notation (high byte - low byte).

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also db_read**

C-Adapter *(mini)*

```
int      db_write
           (int dbno,int dwno, int* amount, void* buffer);
```

Example:
DB10 consisting of 20 data words (DW 0 - DW 19) exists in the PLC. Value 2468 hex is assigned to DW 1 and DW 2.

C-Adapter

```
#include <w95_s7.h>

int    buffer[100];
int    error,dbno,dwno,amount;

:
dbno    = 10;
dwno    = 1;
amount = 2;
buffer[0]  = 0x6824;
buffer[1]  = 0x6824;

error = db_write(dbno,dwno,&amount,buffer);
:
error = unload_tool();
:
```

d_field_read

The function **d_field_read** reads “ amount“ of data bytes from the PLC starting from data byte "no" and stores the read values in a variable field of the PG/PC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also d_field_write**

C-Adapter ***(mini)***

```
int        d_field_read
           (int bstno, int no,int amount,void * buffer);
```

d_field_write

The function **d_field_write** writes "amount“ of bytes from the specified storage area to the PLC, starting from data byte "no".

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also d_field_read**

C-Adapter ***(mini)***

```
int        d_field_write
           (int bstno, int no,int amount,void* buffer);
```

e_field_read

The function **e_field_read** reads “amount” of input bytes from the PLC starting from input byte "no" and stores the read values into a variable field of the PG/PC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also a_field_read, m_field_read**

C-Adapter

```
int       e_field_read (int no,int amount,void * buffer);
```

mb_bittest

This function checks a bit in a specified flag byte and supplies the status of the specified bit via "return_wert" (return value) (= boolean variable).

When bit it set	⇨ return_wert = true or 1
When bit is not set	⇨ return_wert = false or 0

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int       mb_bittest(int mbno,int bitno,char * retwert);
```

mb_resetbit

The function **mb_resetbit** sets a flag in the PLC to 0. It is not checked whether the flag bit exists in the used PLC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also mb_setbit, mb_bittest**

C-Adapter

```
int       mb_resetbit (int mbno,int bitno);
```

mb_setbit

The function **mb_setbit** sets a flag in the PLC to 1. It is not checked whether the flag bit exists in the used PLC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also mb_setbit, mb_bittest**

C-Adapter

```
int        mb_setbit (int mbno,int bitno);
```

m_field_read

The function **m_field_read** reads “amount” of flag bytes from the PLC starting from flag byte "no" and stores the read values in a variable field of the PG/PC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also a_field_read, e_field_read**

C-Adapter

```
int       m_field_read (int no,int amount,void * buffer);
```

m_field_write

The function **m_field_write** writes "amount" of bytes from the specified storage area to the PLC starting from flag byte "no".

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also a_field_write**

C-Adapter

```
int       m_field_write (int no,int amount,void* buffer);
```

t_field_read

The function **t_field_read** reads “amount” of timer words from the PLC starting from timer word "no" and stores the read values in a variable field of the PG/PC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also z_field_read**

C-Adapter

```
int        t_field_read (int no,int amount,void * buffer);
```

z_field_read

The function **z_field_read** reads “amount” of counter words from the PLC starting from counter word "no" and stores the read values in a variable field of the PG/PC.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞**See also t_field_read**

C-Adapter

```
int        z_field_read (int no,int amount,void * buffer);
```

z_field_write

The function **z_field_write** writes “amount” of words to the PLC starting from counter word "no" from the specified storage area.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int        z_field_write (int no,int amount,void* buffer);
```

mix_read

This function enables the user to read mixed data.
The following data can be read and / or written:

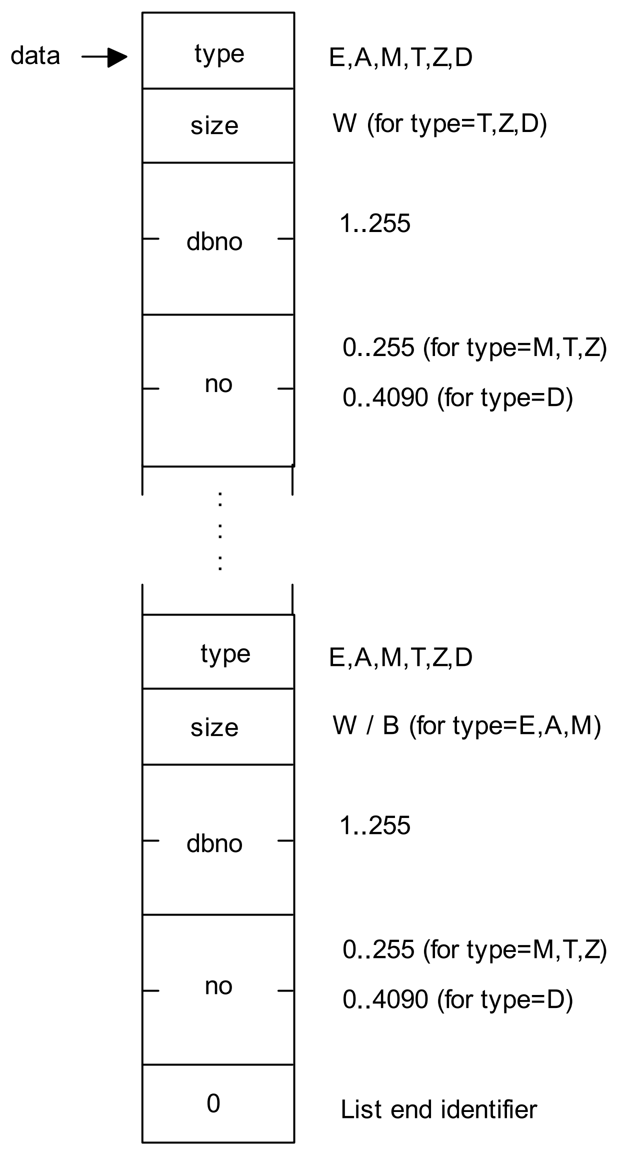
E = Input bytes
A = Output bytes
M = Flag bytes
T = Timer words
Z = Counter words
D = Data from DB

The **mix_read** function reads the data parameterised by “data“ from the PLC and stores the read values in the specified storage area.

A maximum of 20 data may be read. In the event of parameter assignment type = A,E,M size may be set = ”b” or ”w”, parameter assignment type = T,Z,D, size may be set = ”w”.

size = ”w” ⇨ read word and save
size = ”b” ⇨ read byte and save

The read values are saved in sequence. I.e. the user himself must carry out structured processing of the field occupied with the read values. "data" must have the following structure:



C-Adapter

```
int     mix_read (char* data, void* buffer);
```

Example:

Input byte 0, output byte 3 and DW 5 from DB 10 are read and the values are stored in variables e, a and d:

C-Adapter

```
#include <w95_s7.h>


#pragma pack(1)
typedef struct {
               char typ;
               char size;
               unsigned short dbno;
               unsigned short no;
               } data_type;

data_type data[10];
#pragma pack()

char  buffer[100];
char  e,a;
int   d;
int   error;

:
:
data[0].typ = 'e';
data[0].size= 'b';
data[0].no  = 0;
data[1].typ = 'a';
data[1].size= 'b';
data[1].no  = 3;
data[2].typ = 'd';
data[2].size= 'w';
data[2].dbno= 10;
data[2].no  = 5;
data[3].typ = 0;    /* endekennung der liste */

error = mix_read((char*)data,buffer);

e = buffer[0];
a = buffer[1];
d = ((int)buffer[2] << 8) | (int)buffer[3];
:
:
```

mix_write

This function enables the user to write mixed data.
The following data can be read and / or written:

E = Input bytes
A = Output bytes
M = Flag bytes
T = Timer words
Z = Counter words
D = Data in DB

The function **mix_write** overwrites the data in the PLC parameterised by “data“ with the values transferred in “buffer“..

A maximum of 20 data may be written. In the event of parameter assignment of typ = A,E,M, size may be set = ”b” or ”w”, in the event of parameter assignment of type = T,Z,D , size may be set = ”w”.

size = ”w” ⇨ read word and save
size = ”b” ⇨ read byte and save

The values to be written must be entered in sequence. For the structure of "data" see function mix_read.

C-Adapter

```
int      mix_write (char* data, void* buffer);
```

4.1.2 Basic Functions for Data Transfer S7-200

The following data can be read and/or written :

Data Type		CPU 212	CPU214	CPU215	CPU216
Input Bytes	EB	0 - 7	0 - 7	0 - 7	0 - 7
Output Bytes	AB	0 - 7	0 - 7	0 - 7	0 - 7
Flag Bytes	MB	0 - 15	0 - 31	0 - 31	0 - 31
Special Flag Bytes	SM	0 - 45	0 - 85	0 - 199	0 - 199
Variable Memory	VS	0 - 1023	0 - 4095	0 - 5119	0 - 5119
Timers	T	0 - 63	0 - 127	0 - 255	0 - 255
Counters	Z	0 - 63	0 -127	0 - 255	0 -255

For further information on data types and ranges please refer to the Manual STEP 7 - Micro / Programming S7-200.

as200_ag_info

This function reads the issue level of the PLC software and the PC interface as well as the type of PLC and stores them in a storage area of the PG/PLC as an ASCII-String zero-terminated.
The issue levels must be interpreted as integer values, the PLC types as ASCII-values:

buffer[0] - buffer[1]:	⇨ integer value ⇨ issue level firmware
buffer[2] - buffer[3]:	⇨ integer value ⇨ issue level Asic
buffer[4] - buffer[24]:	⇨ PLC-type

If there were no errors, the function supplies a 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞ **Siehe auch as200_ag_zustand**

C-Adapter ***(mini)***

```
int     as200_ag_info (void * buffer);
```

Example:
The issue levels of the PLC software, the PG interface as well as the PLC type are read.

C-Adapter

```
#include <w95_s7.h>

int   error;
adr_table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };
#pragma pack(1)
struct {
   unsigned short plcas
   unsigned short pgas;
   char  mlfb[21];
   } info;
#pragma pack()
:
error = load_tool(1,"S7ONLINE",plc_adr_table);
:
error = as200_ag_info(&info);
:
error = unload_tool();
```

as200_ag_zustand

This function reads the PLC status (RUN or STOP) from the PLC and stores the data in a storage area of the PG/PC.

buffer[0] = 0	⇨	AG is in RUN
buffer[0] <> 0	⇨	AG is in STOP

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

☞See also as200_ag_info

C-Adapter

```
int      as200_ag_zustand (void * buffer);
```

Example:

C-Adapter

```
#include <w95_s7.h>

int  error;
adr_ table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };
char state;
:
error = load_tool(1,"S7ONLINE",plc_adr_table);
:
error = as200_ag_zustand(&state);
if (state == 0); //plc in run
if (state != 0); //plc in stop
:
error = unload_tool();
:
```

as200_e_field_read

as200_a_field_read

as200_m_field_read

as200_sm_field_read

as200_vs_field_read

The functions as200_**X**_field_read read “amount” of bytes from the data area of the PLC starting from "no" and store the read values in a variable field of the PG/PC.

If there were no errors, the function supplies a 0 as the return value, otherwise an error number which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_e_field_read (int no,int amount,void*buffer);

int as200_a_field_read (int no,int amount,void*buffer);

int as200_m_field_read (int no,int amount,void*buffer);

int as200_sm_field_read (int no,int amount,void*buffer);
```

C-Adapter (mini)

```
int as200_vs_field_read (int no,int amount,void*buffer);
```

Example:
Output byte 0..9 is read and saved in ”buffer”.

C-Adapter

```
#include <w95_s7.h>


int   error;
adr_table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };
#pragma pack(1)
unsigned char buffer[10];
#pragma pack()
:
error = load_tool(1,”S7ONLINE”,plc_adr_table);
:
error = as200_a_field_read(0,10,buffer);
:
error = unload_tool();
:
```

as200_a_field_write

as200_m_field_write

as200_sm_field_write

as200_vs_field_write

The functions as200_**X**_field_write write "amount" of bytes from the specified storage area to the PLC starting from "no".

If there were no errors, the function supplies a 0 as the return value, otherwise an error number which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_a_field_write (int no,int amount,void* buffer);

int as200_m_field_write (int no,int amount,void* buffer);

int as200_sm_field_write (int no,int amount,void* buffer);
```

C-Adapter ***(mini)***

```
int as200_vs_field_write (int no,int amount,void* buffer);
```

as200_t_field_read

The functions as200_t_field_read read “amount” of bytes from the data area of the PLC starting from "no" and store the read values in a variable field of the PG/PC.

Attention!

5 bytes are received per timer value, and only 2 bytes contain the requested value (see example)

If there were no errors, the function supplies a 0 as the return value, otherwise an error number which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_t_field_read (int no,int amount,void*buffer);
```

Example:
Timer values 0 and 1 are read and saved in ”T0” and ”T1”.

C-Adapter

```
#include <w95_s7.h>

int   error;
adr_table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };

unsigned char buffer[10];
unsigned short T0,T1;
:
error = load_tool(1,”S7ONLINE”,plc_adr_table);
:
error = as200_t_field_read(0,2,buffer);
T0  = (unsigned short)buffer[4] |
      (unsigned short)buffer[3] << 8;
T1  = (unsigned short)buffer[9] |
      (unsigned short)buffer[8] << 8;
:
error = unload_tool();
:
```

as200_z_field_read

The function as200_z _field_read reads "amount" of counter values from the data area of the PLC starting from "no" and stores the read values in a variable field of the PG/PC.

Attention!

3 bytes are received per counter value, and only 2 bytes contain the requested value (see example)

If there were no errors, the function supplies a 0 as the return value, otherwise an error number which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_z_field_read (int no,int amount,void*buffer);
```

Example:
Counter values 3 and 4 are read and saved in ”Z3” and ”Z4”.

C-Adapter

```
#include <w95_s7.h>

int   error;
adr_table_type plc_adr_table[2] = { {2,0,2,0},{0,0,2,0} };

unsigned char buffer[10];
unsigned short Z3,Z4;
:
error = load_tool(1,”S7ONLINE”,plc_adr_table);
:
error = as200_z_field_read(3,2,buffer);
Z3 = (unsigned short)buffer[2] |
     (unsigned short)buffer[1] << 8;
Z4 = (unsigned short)buffer[5] |
     (unsigned short)buffer[4] << 8;
:
error = unload_tool();
:
```

as200_z_field_write

The function as200_z _field_write writes "amount" of bytes from the specified storage area to the PLC starting from "no".

Attention!

3 bytes are received per counter value, and only 2 bytes contain the requested value

If there were no errors, the function supplies a 0 as the return value, otherwise an error number which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_z_field_write (int no,int amount,void* buffer);
```

as200_mb_bittest

This function checks a bit in a specified flag byte and supplies the status of this bit via "return_wert" (= “return_value") (= boolean variable).

When bit set	➩ return_wert = true or 1
When bit not set	➩ return_wert = false or 0

If there were no errors, the function supplies a 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_mb_bittest(int mbno,int bitno,char * retwert);
```

as200_mb_resetbit

This function sets a flag in the PLC to 0. It is not checked whether the flag bit exists in the used PLC.

If there were no errors, the function supplies a 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_mb_resetbit (int mbno,int bitno);
```

as200_mb_setbit

This function sets a flag in the PLC to 1. It is not checked whether the flag bit exists in the used PLC.

If there were no errors, the function supplies a 0 as the return value, otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int as200_mb_setbit (int mbno,int bitno);
```

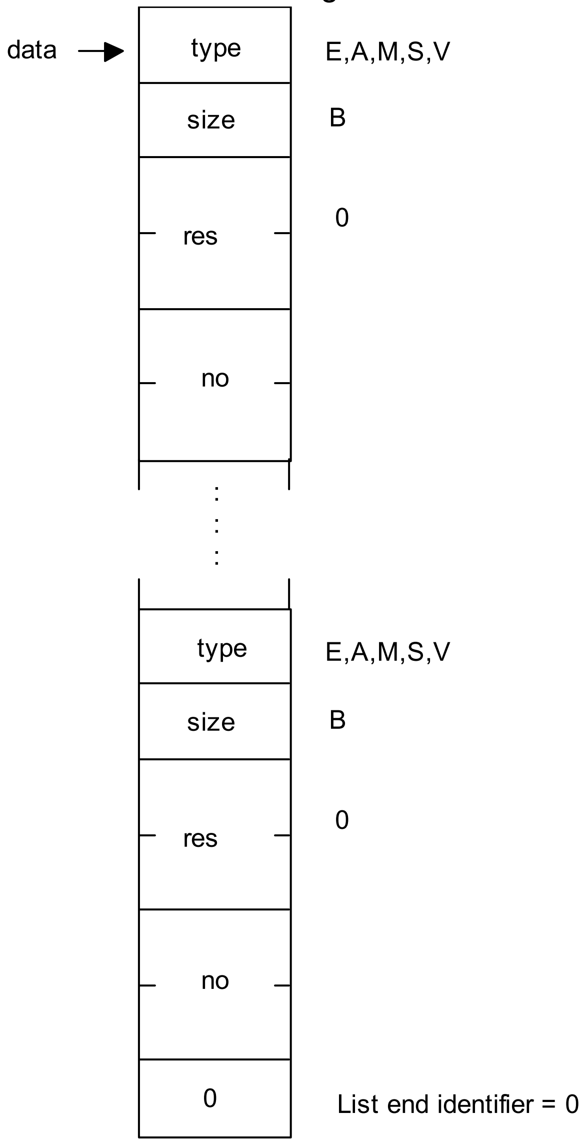
as200_mix_read

The **as200_mix_read** function reads the data parameterised by "data" from the PLC, and stores the read values in the specified storage area.

A maximum of 20 data may be read. In the event of parameter assignmemnt of type = A,E,M size may be set = “b“ or “w“.

type = “E“	➩ read input range
type = “A“	➩ read output range
type = “M“	➩ read flag range
type = “V“	➩ read variable memory
type = “S“	➩ read special flag range

The read values are saved in sequence. I.e. the user himself must carry out structured processing of the field occupied with the read values. "data" must have the following structure:



C-Adapter

```
int     as200_mix_read (void* data, void* buffer);
```

Example:

Input byte 0, output byte 3 and variable memory bytes 5 and 6 are read and the values are stored in variables e, a v5 and v6.and d:

C-Adapter

```
#include <w95_s7.h>


#pragma pack(1)
typedef struct {
               char typ;
               char size;
               int dbno;
               int no;
               } data_type;

data_type data[10];
#pragma pack()

char  buffer[100];
char  e,a,v5,v6;
int   v;
int   error;

:
:
data[0].typ = 'e';  /* eingangsbyte 0 */
data[0].size= 'b';
data[0].no  = 0;
data[1].typ = 'a';  /* ausgangsbyte 3 */
data[1].size= 'b';
data[1].no  = 3;
data[2].typ = 'v';  /* variablen speicher byte 5 */
data[2].size= 'b';
data[2].no  = 5;
data[3].typ = 'v';  /* variablen speicher byte 6 */
data[3].size= 'b';
data[3].no  = 6;
data[4].typ = 0;    /* endekennung der liste */

error = mix_read(data,buffer);

e = buffer[0];
a = buffer[1];
v5= buffer[2];
v6= buffer[3];
:
```

as200_mix_write

The function **as200_mix_write** overwrites the data in the PLC parameterised by "data" with the values transferred in “buffer“.

A maximum of 20 data may be written. In the event of parameter assignment of type = A,E,M,S,V size may be set = “b“.

The values to be written must be entered in sequence. For the structure of "data" see function as200_mix_read.

C-Adapter

```
int     as200_mix_write (void* data, void* buffer);
```

4.2 Enhanced Functions for Data Handling in PG/PC

boolean_byte

The function **boolean_byte** converts eight logical values (PC-display) to a byte. The transferred pointer should point to a char field with the following structure:

char buff[8]	buff[0]	buff[1]	buff[2]	buff[3]	buff[4]	buff[5]	buff[6]	buff[7]
return value	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

C-Adapter

```
char    boolean_byte(char * buff);
```

byte_boolean

The function **byte_boolean** converts a byte to eight logical values (PC-display). The transferred pointer should point to a char field with the following structure:

char buff[8]	buff[0]	buff[1]	buff[2]	buff[3]	buff[4]	buff[5]	buff[6]	buff[7]
Value	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

C-Adapter

```
void    byte_boolean(char wert, char * buff);
```

gp_to_float

The function **gp_to_float** converts an S7 floating point value to a value of the float type (IEEE-Format).

☞**See also float_to_gp**

C-Adapter

```
void      gp_to_float(void * gp, void * ieee);
```

Example:

Assumption:	DBW 0 and DBW 2 = floating point 1,234*10^{-5} or DBW 0 = 374F hex, DBW 2 = 07E5 hex in DB 1.

This program reads 2 data words (DBW 0 and DBW 2), converts the S7 floating point format to IEEE format and makes the value available for processing in the variable "ieee".

C-Adapter

```
#include <komfort.h>
#include <w95_s7.h>

int   error;
int   buffer[100]
int   dbno,dwno,amount;
float ieee;

:
dbno   = 10;
dwno   = 0;
amount = 2;

error = db_read(dbno,dwno,&amount,buffer);
gp_to_float(buffer,&ieee);
:
error = unload_tool();
:
```

float_to_gp

The function **float_to_gp** converts a value of the float type (IEEE-Format) to an S7 floating point value.

☞**See also gp_to_float**

C-Adapter

```
void              floag_to_gp (void * ieee, void * gp);
```

kg_to_float

The function "**kg_to_float** converts an S5 floating point value to a value of the float type (IEEE-Format).

If there were no conversion errors, the function supplies 0 as the return value, otherwise 1.

☞**See also float_to_kg**

C-Adapter

```
int        kg_to_float(void * kg, void * ieee);
```

float_to_kg

The function **float_to_kg** converts a value of the float type (IEEE-Format) to an S5 floating point value.

If there were no conversion errors, the function supplies 0 as the return value, otherwise 1.

☞**See also kg_to_float**

C-Adapter

```
int        floag_to_kg (void * ieee, void * kg);
```

kf_integer

The function **kf_integer** swaps the high byte and the low byte of a transferred 16-bit value and returns the new value.

C-Adapter

```
unsigned short  kf_integer (unsigned short);
```

testbit

The function **testbit** checks whether a specified bit is set in a byte variable. The byte variable and the bit number are transferred to the function in the form of parameters.

Return value = TRUE or 1	⇨	Bit is set
Return-value = FALSE or 0	⇨	Bit is not set

C-Adapter

```
char    testbit (char value, char bitno);
```

error_message

This function supplies the approprioate error text relating to an error message in the form of a zero terminated character string. For this purpose the ERROR.DAT file is read when the function is first called and the texts are stored in a file.
When transferring error number 0 the file name of the error text file to be loaded can be transferred in “buffer“. If no valid file name was transferred or a ZERO pointer was transferred, the ERROR.DAT file is read in the current directory. Therefore it must be ensured that the ERROR.DAT file exists and is in the same directory as the program.

The error texts are stored in English. Should you require the error texts in German, you must rename the **ERROR.GER** file to **ERROR.DAT,** and copy this file into the directory of your application, or load this file explicitly.
.
A maximum of 100 error texts can be stored.

Note:
We recommend calling the error_message function shortly after program start by means of error_no = 0 to load the ERROR.DAT file. This ensures almost consistent processing time for further calls of this function.

Structure error text file:
[Error number as ascii hex]:[error text]
e.g.:
0207:data segment cannot be disabled
0302:block too small DW does not exist
...

If there were no errors, the function supplies 0 as the return value.

Return-Value = 1: ERROR.DAT file does not exist or cannot be opened.

Return-Value = 2: Error when reading the ERROR.DAT file.

Return-Value = 3: Incorrect call of the ERROR.DAT error text file.

Return-Value = 4: No error text exists for this error number.

Return-Value = 5: Too many error texts in ERROR.DAT

C-Adapter

```
int       error_message (int no,char * buffer);
```

Example:
This program supplies the English error text for an error number. For instance, if data block 10 did not exist in the PLC, the error message "Block not available" or "Baustein nicht vorhanden" would be entered in the “message” variable.

C-Adapter

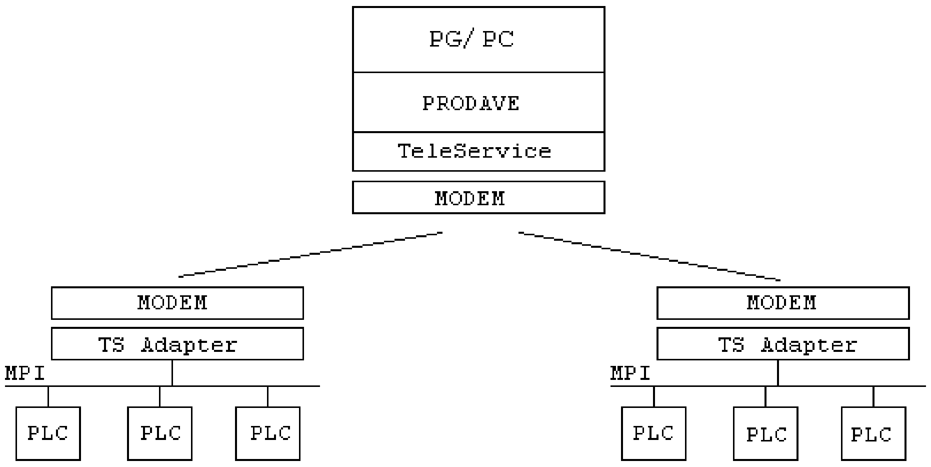
```
#include <komfort.h>
#include <w95_s7.h>

int   buffer[100];
int   error;
int   amount;
char  message[100];

:
:
error = error_message(0,"error.eng");
:
amount = 45;

error = db_read(10,5,&amount,buffer);
if (error != 0)
   error = error_message(error,message);
:
error = unload_tool();
:
:
```

4.3 TeleService Funcions

The Teleservice functions enable the user to establish and close remote connections between PG/PCs and TS Adaptors. The conversion of data between the public telephone network / modem and the MPI interface is carried out by the TS Adaptor hardware.

The pre-requisite for operation of PRODAVE TeleService functions is the installation of software package TeleService.

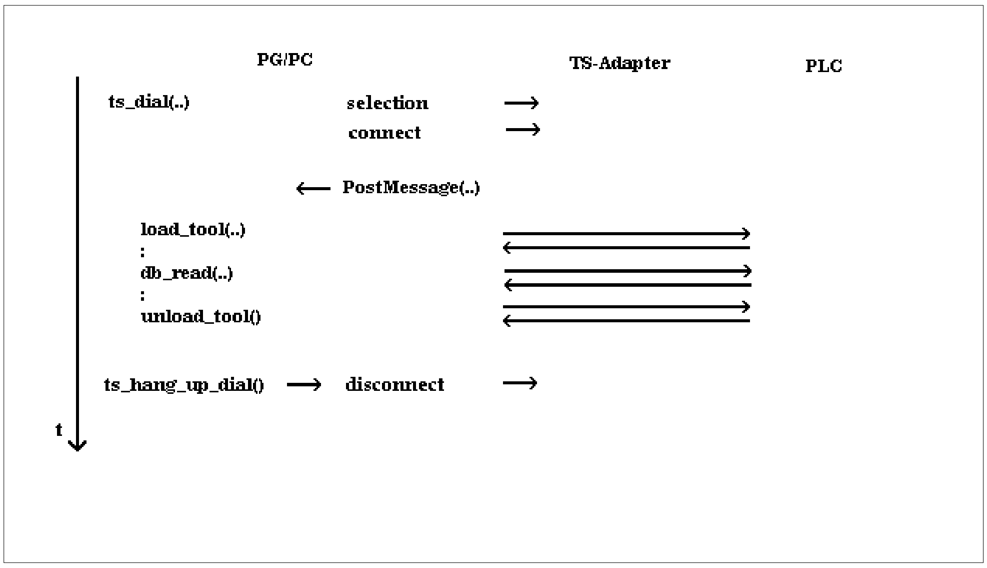
Parameterisation of the TS-Adaptor is carried out using the SIMATIC Software TeleService.



When the TeleService functions are in use it is important to note that the PG/PC interface in system control the module parameter assignment is set to TS-Adaptor

A modem connection is basically only estalblished with the TeleService functions. There are two possibilities in order to establish the connections

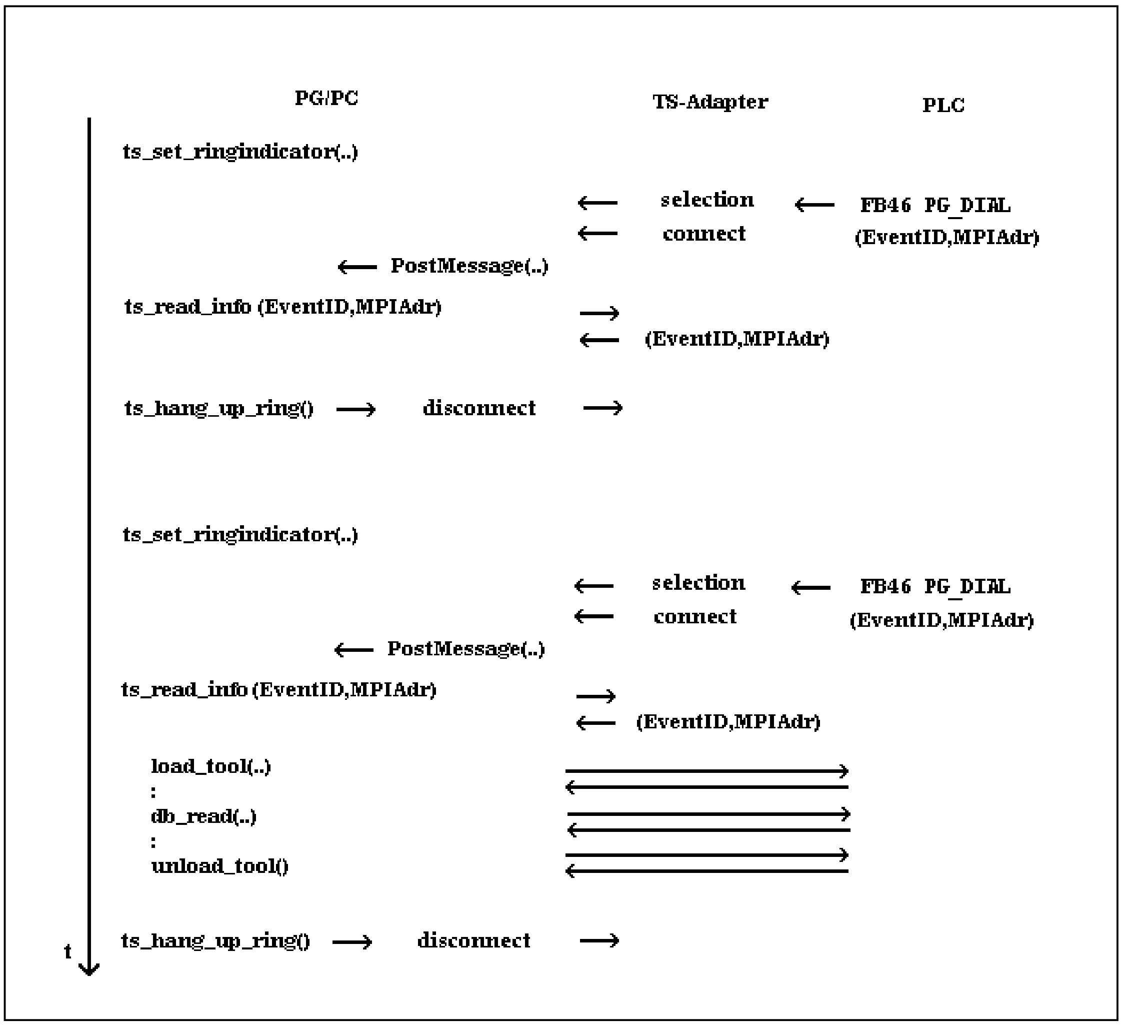
- o actively by the PG/PC using the **ts_dial** function



When access protection is activated the TS-Adaptor checks the specified password. Providing the password is correct, the connection is established either immediately or after re-dialling from the TeleService Adaptor (depending on TeleService parameterisation).

The function ts_dial returns the connection status and / or in the event of an asynchronous call of function ts_dial the message TS_CONNECTED (connection established) or TS_DISCONNECTED (connection not established) is sent to the specified window only after the connection is established or if the TeleService Adaptor replies with error (e.g. password not correct).

- actively by the PLC, in this instance it is required to activate a ring indication in the PG/PC using the **ts_set_ringindicator** function.



After the TeleService connection has been established successfully, the user can establish a connection to the PLC connected to the TS adaptor using the **load_tool** function. It is possible to use any amount of PRODAVE functions. This is followed by closing the connection to the PLC using the **unload_tool** function and closing the TeleService connection by means of **ts_hang_up_dial** or **ts_hang_up_ring**.

Note:

After establishing a **remote connection** this **remains active** until the user program calls the **ts_hang_up_dial** and / or **ts_hang_up_ring** functions!

4.3.1 General Commments on the Modem Connection

The Modem parameters such as dial parameters, location parameters etc. must be specified in the WINDOWS system control / modems.

It is possible to have 1 telephone connection only open at one time.

4.3.2 Active Telephone Dialling from the PLC

The PLC has the facility to establish a TeleService connection via the TS Adaptor. This is carried out by calling the function block 46 PG_DIAL.

For further information please refer to the Product Description TS Adaptor and SIMATIC TeleService.

ts_dial

The function **ts_dial** dials a remote station via the modem and establishes the connection to the TS-Adaptor. When access protection is activated the password is checked by the TS-Adaptor and the TS-Adaptor may ring back, if required.

The following parameters are transferred:

- **ModemName**
  Name of Modem to be used, can be selected in system control / modems

- **Location**
  Name of Modem location, can be selected in system control / modem / dial parameters

- **TelNo**
  Telephone number, which is dialled by the connected modem.
  The telephone number must be transferred in canonical format:
  + CountryCode Space [area code Space] Tel.Nr. | SubAddress(ISDN) ^ Name(ISDN)
  z.B.: +49 0711 137 3969
  maximum amount of characters to be input is 31.

- **UserName**
  Specifiy UserName parameterised in TS Adaptor to be called.
  Maximum amount of characters to be input is 8.

- **Password**
  Specify password parameterised in TS Adaptor to be called.
  Maximum amount of characters to be input is 8.

- **WindowHandle**
  Here it is possible to transfer a window handle. If the handle is valid, the **ts_dial** function is processed asynchronously, i.e. the calling application is advised as to whether the connection has been estalblished positively or negatively by means of a Windows message.
  If the WindowHandle NULL (ZERO) is transferred, the function is processed synchronously, i.e. the function does not return until after the connection has been established or the timeout has elapsed.

- **Message**
  Message which is sent to the window when the connection has been established or the timeout has elapsed.

- **wParam**
  Parameters for the message.

- **res1**
  reserved, must be set to NULL.

In the event of a valid WindowHandles messages with PostMessage(Message,wParam,lParam) are transmitted. In this instance lParam High-Word is the error code, lParam Low-Word has the following meaning:

TS_CONNECTED = 1	connection established
TS_DISCONNECTED = 3	connection not established

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int     ts_dial ( char *  ModemName,
                  char *  Standort,
                  char *  TelNo,
                  char *  UserName,
                  char *  Password,
                  HANDLE  WindowHandle,
                  UINT    Message,
                  WPARAM  wParam,
                   char *  res1);
```

Example:

A remote station is called. If the connection to the TS adaptor has been established this is followed by the connection of a controller connected to the TS adaptor:

C-Adapter

```
#include <w95_s7.h>


#pragma pack(1)
#pragma pack()

char   * ModemName = "Standardmodem";
char   * Standort  = "Standardstandort";
char   * TelNo = "+49 00911 8957101";
char   * UserName = "ADMIN";
char   * Password = "admin";

int    error;

:
:
error = ts_dial(ModemName,Standort,TelNo,
                UserName,Password,NULL,0,0,NULL);
if (error == 0)
   {
   error = load_tool(...);
   :
   :
   :
   error = unload_tool();
   error = ts_hang_up_dial();
   }
:
:
```

ts_hang_up_dial

The function **ts_hang_up_dial** interrupts the current connection or an asynchronous dialling process currently running.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int     ts_hang_up_dial (void);
```

ts_set_ringindicator

The function **ts_set_ringindicator** initialises the sub-ordinate system for receiving calls, establishing the connection and reporting (Ring indication). A connection which is being established because of an incoming call is signalled by means of a Windows message.

The following parameters are transferred:

- **ModemName**
  Name of the modem to be used for the ring indication, can be selected in system control / modem

- **NumberOfRings**
  Number of rings until the modem replies.

- **WindowHandle**
  Here it is possible to transfer a window handle. If the handle is valid, the calling application is advised as to whether the connection has been estalblished positively or negatively by means of a Windows message.
  If the WindowHandle NULL (ZERO) is transferred, it is possible to recognize a successful connection by means of the cyclic structure of the **ts_read_info** function.

- **Message**
  Message which is sent to the window when the connection has been established.

- **wParam**
  Parameters for the message.

- **res1**
  reserved, must be set to NULL.

In the event of a valid WindowHandle messages with PostMessage(Message,wParam,lParam) are transmitted. In this instance lParam has the following meaning:

TS_CONNECTED = 1	connection established

In order to end the ring indication call the function with **ModemName=NULL (ZERO)**.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int      ts_set_ringindicator (
                   char *  ModemName,
                   int     NumberOfRings
                   HANDLE  WindowHandle,
                   UINT    Message,
                   WPARAM  wParam,
                   char *  res1);
```

ts_read_info

The function **ts_read_info** supplies informationen on the alarm triggering station.

The following parameters are transferred:

- **EventId**
  Pointer to a field 16 Bytes long. Information on the alarm triggering station is entered here.
  See Product Description SIMATIC TeleService, FB PG-DIAL.

- **MpiAdr**
  Pointer to a Byte. The MPI address of the alarm triggering station is entered here.

If no connection is established via the ring indication the function supplies error number 0336 Hex.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int      ts_read_info (    void * EventID,
                           unsigned char * MpiAdr);
```

ts_hang_up_ring

The function **ts_hang_up_ring** interrupts the connection established by the TS adaptor.

If there were no errors, the function supplies 0 as the return value otherwise an error message which can be evaluated in accordance with the error table (see error_message).

C-Adapter

```
int      ts_hang_up_ring (void);
```

5. Demonstration Programs

5.1 Demonstration Programs for the PC

After successful installation **demonstration programs for the PC** can be found under the path

- \SIEMENS\PRODAVE_S7\SAMPLE for PRODAVE S7
- \SIEMENS\PRODAVE_S7_MINI\SAMPLE for PRODAVE S7 mini

In accordance with these program examples we show in a clearly visible format how the PRODAVE functions can be used. To ensure that the examples are not overloaded we have realised only a few of the functions.
It is important to note that the two programs do not profess to be complete. They merely serve to provide assistance when programming your application.

The demonstration programs operate on the principle that process data traffic to a PLC is based on **Address = 2** and **Slot no = 2**!

When connecting an S7-400 with double wide power supply module set **Slot no = 3**.

Calling the Demonstration Programs for Windows 95/98/NT/ME/2000/XP:

- Insert the appropriate data link cable PLC - PG/PC into the PG interface on the PLC and into the MPI interface (and/or COM-Port when using the PC-Adaptor- cable) of the PG/PC.

- Configure the used PG/PC interface using the STEP7 tool (S7EPATSX.EXE).
  The access point of application **”S7ONLINE”** must be linked to the used module parameterisation.

- In the event of a data link to S7-200 it is required to dial the relevant module parameterisation with the suffix (PPI)

- Start Windows 95 again in to ensure the configuration is accepted.

- Start the demonstration program in the PRODAVE program group.

- Select the **load_tool** menu and specify the parameters (address, slot number, segment ID and rack number) of the destination system.

6. Appendix

6.1 Error Texts

You may add your own error texts in the ERROR.DAT file to the ones listed below. See function "error_message".

Error Messages:

0000	:	** ERROR.DAT = error text file for PRODAVE S7 **
00CA	:	no resources available
00CB	:	configuration error
00CD	:	illegal call
00CE	:	module not found
00CF	:	driver not loaded
00D0	:	hardware fault
00D1	:	software fault
00D2	:	memory fault
00D7	:	no message
00D8	:	storage fault
00DB	:	internal timeout
00E1	:	too many channels open
00E2	:	internal fault
00E7	:	hardware fault
00E9	:	sin_serv.exe not started
00EA	:	protected
00F0	:	scp db file does not exist
00F1	:	no global dos storage available
00F2	:	error during transmission
00F2	:	error during reception
00F4	:	device does not exist
00F5	:	incorrect sub system
00F6	:	unknown code
00F7	:	buffer too small
00F8	:	buffer too small
00F9	:	incorrect protocol
00FB	:	reception error
00FC	:	licence error

0101	:	connection not established / parameterised
010A	:	negative acknowledgement received / timeout error
010C	:	data does not exist or disabled
012A	:	system storage no longer available
012E	:	incorrect parameter
0132	:	no memory in DPRAM
0201	:	incorrect interface specified
0202	:	maximum amount of interfaces exceeded
0203	:	PRODAVE already initialised
0204	:	wrong parameter list
0205	:	PRODAVE not initialised
0206	:	handle cannot be set
0207	:	data segment cannot be disabled
0300	:	initialisiation error
0301	:	initialisiation error
0302	:	block too small, DW does not exist
0303	:	block limit exceeded, correct amount
0310	:	no HW found
0311	:	HW defective
0312	:	incorrect config param
0313	:	incorrect baud rate / interrupt vector
0314	:	HSA parameterised incorrectly
0315	:	MPI address error
0316	:	HW device already allocated
0317	:	interrupt not available
0318	:	interrupt occupied
0319	:	sap not occupied
031A	:	no remote station found
031B	:	internal error
031C	:	system error
031D	:	error buffer size
0320	:	hardware fault
0321	:	DLL function error
0330	:	version conflict
0331	:	error com config
0332	:	hardware fault
0333	:	com not configured
0334	:	com not available
0335	:	serial drv in use
0336	:	no connection
0337	:	job rejected
0380	:	internal error
0381	:	hardware fault
0382	:	no driver or device found
0384	:	no driver or device found

03FF	:	system fault
0800	:	toolbox occupied
4001	:	connection not known
4002	:	connection not established
4003	:	connection is being established
4004	:	connection broken down
8000	:	function already actively occupied
8001	:	not allowed in this operating status
8101	:	hardware fault
8103	:	object access not allowed
8104	:	context is not supported
8105	:	invalid address
8106	:	type (data type) not supported
8107	:	type (data type) not consistent
810A	:	object does not exist
8301	:	memory slot on CPU not sufficient
8404	:	grave error
8500	:	incorrect PDU size
8702	:	address invalid
D201	:	syntax error block name
D202	:	syntax error function parameter
D203	:	syntax error block type
D204	:	no linked block in storage medium
D205	:	object already exists
D206	:	object already exists
D207	:	block exists in EPROM
D209	:	block does not exist
D20E	:	no block available
D210	:	block number too big
D241	:	protection level of function not sufficient
D406	:	information not available
EF01	:	incorrect ID2
FFFB	:	TeleService Library not found
FFFE	:	unknown error FFFE hex
FFFF	:	timeout error. Check interface

TeleService Error Messages:

0048	:	error during connection
4350	:	not implemented
4360	:	timeout
8001	:	no memory
8305	:	error during access to Registry
8306	:	adaptor in direct operation
8FFF	:	internal error
8305	:	error during access to Registry
4501	:	incorrect parameter, modem or location error
4502	:	no further entries
4503	:	modem function not sufficient
4504	:	transferred string too long
4510	:	adaptor in Modem operation
4540	:	alarm already allocated
4541	:	alarm not used
4580	:	login error user name
4581	:	login error password
A0CE	:	busy
A0CF	:	partner not responding
A0D4	:	connection not available
A0D5	:	no dial tone

6.2 Used Abbreviations

PLC	Programmable Logic Controller
CP	Communications Processor
CPU	Central-Processing-Unit
DB	Data Block
DLL	Dynamic Link Library
MPI	Multi Point Interface
PC	Personal Computer
PG	Programming Unit
PRODAVE	Process Data Traffic

6.3 Literature and Ordering Data

6.3.1 Modules

TS-Adaptor V5
Part No.: 6ES7-972-0CA32-0XA0

TS-Adaptor V5.1
Part No.: 6ES7-972-0CA33-0XA0

PC-Adaptor
Part No.: 6ES7-972-0CA23-0XA0

PC/PPI Cable
Part No.: 6ES7-901-3BF00-0XA0

CP5511 PROFIBUS PCMCIA-Interface
Part No.: 6GK1-551-1AA00

CP5611 PROFIBUS PCI-Interface
Part No.: 6GK1-561-1AA00

6.3.2 PRODAVE Overview

PRODAVE S7 mini Win95/98/NT/ME/2000/XP including manual
Part No.: 6ES7-807-3BA00-0YA0

PRODAVE S7 Win95/98/NT/ME/2000/XP including manual
Part No.: 6ES7-807-4BA00-0YA0