SIEMENS

SIMATIC

PRODAVE MPI V6.0

Manual

Preface, Contents

Introduction 1

Description 2

Operation 3

PRODAVE MPI V6.0 Functions 4

Demonstration Programs 5

Appendix A

Edition 05/2005
A5E00417150-01

Safety Guidelines

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 to property damage only have no safety alert symbol. The 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
with a safety alert symbol indicates that minor personal injury can result if proper precautions are not taken.

Caution
without a safety alert symbol indicates that property damage can result if proper precautions are not taken.

Attention
indicates that an unintended result or situation can occur if the corresponding notice is not taken into account.

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 device/system may only be set up and used in conjunction with this documentation. Commissioning and operation of a device/system may only be performed by **qualified personnel.** Within the context of the safety notices in this documentation qualified persons are defined as persons who are authorized to commission, ground and label devices, systems and circuits in accordance with established safety practices and standards.

Prescribed Usage

Note the following:

Warning
This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.
Correct, reliable operation of the product requires proper transport, storage, positioning and assembly as well as careful operation and maintenance.

Trademarks

All names identified by ® are registered trademarks of the 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.

Siemens AG
Automation and Drives
Postfach 4848, 90327 Nuremberg, Germany

Siemens Aktiengesellschaft A5E00417150-01

Preface

Purpose of the Manual

This manual gives you a complete overview of the PRODAVE MPI V6.0 functions.

This manual is intended for those responsible for configuring, commissioning, and servicing automation systems.

Required Basic Knowledge

You require a general knowledge in the field of automation engineering to be able to understand this manual.

In addition, you should know how to use computers or devices with similar functions (e.g programming devices) under Windows 2000 or XP operating systems.

Where is this Manual valid?

This manual is valid for the software package PRODAVE MPI V6.0.

Further Support

If you have any technical questions, please get in touch with your Siemens representative or agent responsible.

You will find your contact person at:

http://www.siemens.com/automation/partner

You will find a guide to the technical documentation offered for the individual SIMATIC Products and Systems here at:

http://www.siemens.com/simatic-tech-doku-portal

The online catalog and order system is found under:

http://mall.automation.siemens.com/

Training Centers

Siemens offers a number of training courses to familiarize you with the SIMATIC S7 automation system. Please contact your regional training center or our central training center in D 90327 Nuremberg, Germany for details:

Telephone: +49 (911) 895-3200.

Internet: http://www.sitrain.com

Technical Support

You can reach the Technical Support for all A&D products

- Via the Web formula for the Support Request
  http://www.siemens.com/automation/support-request
- Phone: + 49 180 5050 222
- Fax: + 49 180 5050 223

Additional information about our Technical Support can be found on the Internet pages http://www.siemens.com/automation/service

Service & Support on the Internet

In addition to our documentation, we offer our Know-how online on the internet at:

http://www.siemens.com/automation/service&support

where you will find the following:

- The newsletter, which constantly provides you with up-to-date information on your products.
- The right documents via our Search function in Service & Support.
- A forum, where users and experts from all over the world exchange their experiences.
- Your local representative for Automation & Drives.
- Information on field service, repairs, spare parts and more under "Services".

Contents

1 Introduction 1-1

1.1 Basic Functions 1-2
1.1.1 Functions for Data Transfer to S7 300/400 1-2
1.1.2 Functions for Data Transfer to S7 200 1-3
1.2 Functions for Data Handling in PG/PC 1-4

2 Description 2-1

2.1 Operating Principle of PRODAVE 2-1
2.2 Using the Programming Language Adapter 2-1
2.3 Requirements 2-1
2.4 Connecting the PG/PC to the PLC 2-2
2.4.1 Driver under Windows 95/98/NT 2-2

3 Operation 3-1

3.1 Installing PRODAVE MPI V6.0 3-1
3.1.1 Installing PRODAVE MPI V6.0 under Windows 95/98/NT/ME/2000/XP 3-1
3.2 Scope of Supply of PRODAVE MPI V6.0 3-2
3.2.1 PRODAVE MPI V6.0 for Windows 95/98/NT/ME/2000/XP 3-2
3.2.2 PRODAVE MPI V6.0 Mini for Windows 95/98/NT/ME/2000/XP 3-3
3.3 Working with PRODAVE 3-4
3.3.1 Notes on S7-200 3-4
3.3.2 Notes on S7-300/400 3-4
3.4 Differences between S5 and S7 3-5
3.5 Linking to Standard Tools 3-7
3.5.1 PRODAVE under Delphi (32-Bit) Example 3-7
3.5.2 PRODAVE under Access (32-Bit) Example 3-8
3.5.3 PRODAVE under Visual Basic (32-Bit) Example 3-8

4 PRODAVE MPI V6.0 Functions 4-1

4.1 Basic Functions 4-3
4.1.1 load_tool 4-3
4.1.2 unload_tool 4-5
4.1.3 new_ss 4-6
4.2 Functions for Data Communication Traffic to the S7 300/400 4-7
4.2.1 ag_info 4-7
4.2.2 ag_zustand 4-9
4.2.3 db_buch 4-10
4.2.4 db_read 4-11
4.2.5 d_field_read 4-13
4.2.6 e|a|m_field_read 4-14
4.2.7 t|z_field_read 4-16
4.2.8 mix_read 4-17
4.2.9 db_write 4-21
4.2.10 d_field_write 4-23
4.2.11 a|m_field_write 4-24
4.2.12 z_field_write 4-25
4.2.13 mix_write 4-26
4.2.14 mb_setbit 4-30

4.2.15 mb_resetbit....4-31
4.2.16 mb_bittest....4-32
4.3 Functions for Data Communication Traffic to the S7 200....4-33
4.3.1 as200_ag_info....4-33
4.3.2 as200_ag_zustand....4-34
4.3.3 as200_e|a|m|sm|vs_field_read....4-34
4.3.4 as200_t_field_read....4-37
4.3.5 as200_z_field_read....4-39
4.3.6 as200_mix_read....4-41
4.3.7 as200_a|m|sm|vs_field_write....4-44
4.3.8 as200_z_field_write....4-46
4.3.9 as200_mix_write....4-47
4.3.10 as200_mb_setbit....4-50
4.3.11 as200_mb_resetbit....4-51
4.3.12 as200_mb_bittest....4-52
4.4 Comfort Functions....4-53
4.4.1 error_message....4-53
4.4.2 kg_to_float....4-55
4.4.3 float_to_kg....4-56
4.4.4 gp_to_float....4-57
4.4.5 float_to_gp....4-58
4.4.6 testbit....4-59
4.4.7 byte_boolean....4-60
4.4.8 boolean_byte....4-61
4.4.9 kf_integer....4-62
4.4.10 swab_buffer....4-63
4.4.11 copy_buffer....4-64
4.4.12 USHORT_2_bcd....4-65
4.4.13 bcd_2_USHORT....4-66
4.5 Teleservice Functions....4-67
4.5.1 ts_dial....4-67
4.5.2 ts_hang_up_dial....4-69
4.5.3 ts_set_ringindicator....4-70
4.5.4 ts_read_info....4-72
4.5.5 ts_hang_up_ring....4-73
4.5.6 ts_get_modem_name....4-74

5 Demonstration Programs 5-1

5.1 Demonstration Programs for the PC....5-1

A Appendix A-1

A.1 Error Texts....A-1
A.2 Used Abbreviations....A-4

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 bases, 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 MPI V6.0 will offer the solution. PRODAVE MPI V6.0 offers 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/XP.

Via these combined functions the process data traffic between PLC and PG/PC is established by PRODAVE MPI V6.0 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 MPI V6.0 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 MPI V6.0 and to enable you to familiarize yourself with it, we supply several demonstration programs as examples. These functions are fully operational and are available in source code (see chapter "Demonstration Programs").

PRODAVE MPI V6.0 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

- initialize and de-initialize 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 the 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 (=optional software package to STEP 7) for the linking of SIMATIC S7 controllers (PLCs) via the public telephone network.

- Dial a station and / or a TS adapter (ts_dial)
- Close a TeleService connection (ts_hang_up_dial)
- Initialize the system for call recognition (ts_set_ringindicator)
- Read information on alarm triggering station (ts_read_info)
- Close a TeleService connection (ts_hang_up_ring)

2 Description

2.1 Operating Principle of PRODAVE

Using the programming package PRODAVE MPI V6.0 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 MPI V6.0 consists basically of two parts:

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

PRODAVE MPI V6.0 offers the adapter 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 want to read data from the PLC or write data to the PLC using a programming language, you will only require the adapter and its functions.

2.2 Using the Programming Language Adapter

A detailed description of the available functions for the various programming languages of this manual can be found in chapter "PRODAVE MPI V6.0 Functions".

2.3 Requirements

PRODAVE MPI V6.0 operates with the following PLC types: S7-200, S7-300, S7-400, M7 and C7 from the S7 series.

Software Requirement:

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

Hardware Requirement:

PRODAVE MPI V6.0
PRODAVE MPI V6.0 Mini
Simatic PG or AT compatible industrial PC with 64MB main memory and MPI-ISA interface, CP5511, CP5611, CP 5512 or PC Adapter.

2.4 Connecting the PG/PC to the PLC

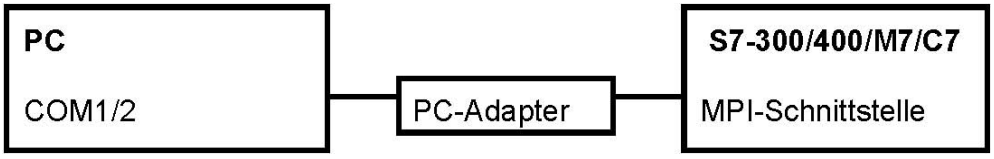
2.4.1 Driver under Windows 95/98/NT

You can connect the PG/PC to the PLC using the following components:

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

PC S7-300/400/M7/C7

MPI-ISA MPI interface CP5511/5512/5611 **S7-200** PPI interface



Installation and set-up of the required hardware is carried out via the STEP 7 tool **Setting the PG/PC interface**, which is available in the control panel after successful installation.

3 Operation

3.1 Installing PRODAVE MPI V6.0

3.1.1 Installing PRODAVE MPI V6.0 under Windows 95/98/NT/ME/2000/XP

The installation of PRODAVE MPI V6.0 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 MPI V6.0 for Windows 95/98/NT/ME/2000/XPPRODAVE DLL and demonstration program for Windows95/98/NT/ME/2000/XP. STEP7 Driver for Windows 95/98/NT/ME/2000/XP
- Documentation.

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

The drivers to be used can be loaded, assigned parameters and linked into the system by means of the STEP 7 tool **Setting the PG/PC-interface** (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 of PRODAVE MPI V6.0

3.2.1 PRODAVE MPI V6.0 for Windows 95/98/NT/ME/2000/XP

The following PRODAVE components are available after a successful installation:

SIEMENS\PRODAVE_S7\INCLUDE\

- W95_S7 .H = Headerfile für 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\LIB\

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

SIEMENS\PRODAVE_S7\SAMPLE_VC\

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

SIEMENS\PRODAVE_S7\SAMPLE_VB\

- VBDEMO .VBP = Visual Basic project file
- ERROR .DAT = error text file
- VBDEMO .EXE = demonstration program
- VBDEMO .BAS = Function declarations for VB
- VBDEMO .FRM = Forms
- 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 MPI V6.0 Mini for Windows 95/98/NT/ME/2000/XP

The following PRODAVE components are available after a successful installation:

SIEMENS\PRODAVE_S7_MINI\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\LIB\

W95_S7M .LIB = import library for PRODAVE-DLL

KOMFORT .LIB = import library for enhanced DLL

SIEMENS\PRODAVE_S7_MINI\SAMPLE_VC_MINI\

DEMO .EXE = demonstration program

DEMO .DSP = Visual C project file

DEMO .C = source code demonstration program

ICON1 .ICO = 32 x 32 icon

DEMO .RC = resource code demonstration program

ERROR .DAT = file with German error texts

RESOURCE.H = header file demonstration program

SIEMENS\PRODAVE_S7_MINI\SAMPLE_VB_MINI\

VBDEMO .VBP = visual basic project file

ERROR .DAT = error text file

VBDEMO .EXE = demonstration program

VBDEMO .BAS = Function declarations for VB

VBDEMO .FRM = Forms

ERROR .FRM

INFO .FRM
LOAD .FRM
READ_FRM .FRM
STATUS .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 programming language and the function calls are used in the form listed in chapter "PRODAVE MPI V6.0 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 configured in the **load_tool** function.

The connection is **initialized** 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 adapter (see also section "Basic Functions for Data Transfer to S7-200"). When you want to end your program, it is required to **de-initialize** the connections by means of the **unload_tool** function.

3.3.2 Notes on S7-300/400

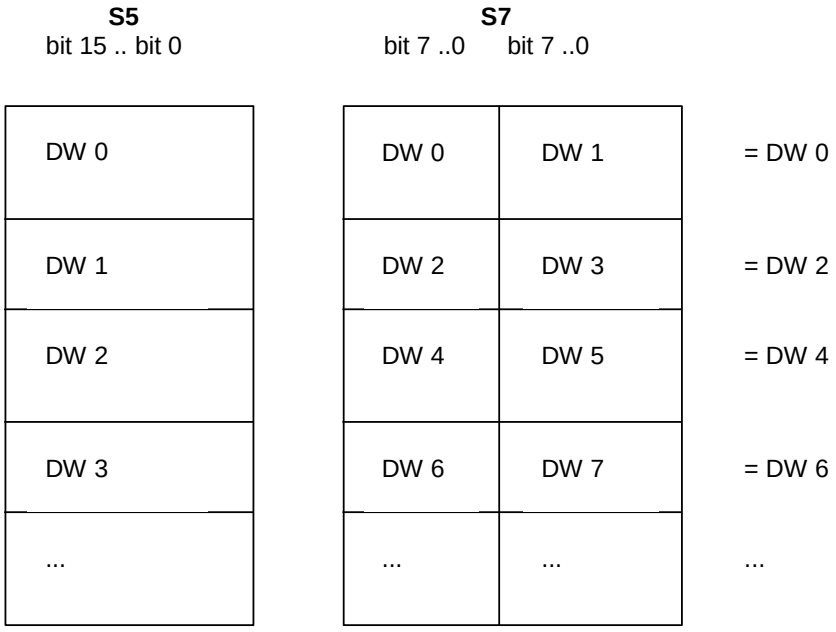
The obligatory start of each user program is the **initialization** of the connections by calling the function **load_tool**. This is followed by the user specific part, where you can call any amount of PRODAVE functions (with the exception of the as200_.... functions) from the adapter. When you want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

When developing 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-initialized by calling the adapter 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 adapter 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 adapter 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. The error text file is loaded when calling this function for the first time.

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	File Type	Start Value	Comment
		STRUCT		
	DW	ARRAY[0 .. 255]		
		WORD		
		END STRUCT		

Example for accessing the variable in the 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; {slot no}
plc_adr_table[3] := 0; {rack no}
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" (ByVal no As Byte, ByVal name As String, ByVal adr As String) As Long

Declare Function db_read Lib " w95_s7" (ByVal dbno As Long, ByVal dwno As Long, amount As Long, buffer As Integer) As Long

Declare Function unload_tool Lib " w95_s7" () 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 PRODAVE MPI V6.0 Functions

Version	**PRODAVE MPI V6.0** **up to V5.6**	**PRODAVE MPI Mini V6.0** **up to V5.6**	**New Function Name in V6.0**
Library	**W95_S7. DLL** **KOMFORT.DLL**	**W95_S7M.DLL** **KOMFORT.DLL**	
Header	**W95_S7. H** **KOMFORT.H**	**W95_S7M.H** **KOMFORT.H**	
Basic Functions			
load_tool: MPI/Profibus	x	x	LoadConnection_ex6
unload_tool: MPI/Profibus or ISO protocol	x	x	UnloadConnection_ex6
new_ss: MPI/Profibus or ISO protocol	x	x	SetActiveConnection_ex6
Functions for data communication traffic to the S7 300/400			
ag_info	x	x	as_info_ex6
ag_zustand	x		as_zustand_ex6
db_buch	x		db_buch_ex6
db_read	x	x	db_read_ex6
d_field_read	x		field_read_ex6
e_field_read	x		
a_field_read	x		
m_field_read	x		
t_field_read	x		
z_field_read	x		
mix_read	x		
db_write	x	x	db_write_ex6
d_field_write	x		field_write_ex6
a_field_write	x		
m_field_write	x		
z_field_write	x		
mix_write	x		
mb_setbit	x		mb_setbit_ex6
mb_resetbit	x		
mb_bittest	x		mb_bittest_ex6
Functions for data communication traffic to the S7 200			
as200_ag_info	x	x	as200_as_info_ex6
as200_ag_zustand	x		as200_as_zustand_ex6
as200_e_field_read	x		as200_field_read_ex6
as200_a_field_read	x		

as200_m_field_read	x		
as200_sm_field_read	x		
as200_vs_field_read	x	x	
as200_t_field_read	x		
as200_z_field_read	x		
as200_mix_read	x		
as200_a_field_write	x		as200_field_write_ex6
as200_m_field_write	x		
as200_sm_field_write	x		
as200_vs_field_write	x	x	
as200_z_field_write	x		
as200_mix_write	x		
as200_mb_setbit	x		as200_mb_setbit_ex6
as200_mb_resetbit	x		
as200_mb_bittest	x		as200_mb_bittest_ex6
Comfort Functions			
error_message	x	x	GetErrorMessage_ex6
kg_to_float	x	x	kg_2_float_ex6
float_to_kg	x	x	float_2_kg_ex6
gp_to_float	x	x	gp_2_float_ex6
float_to_gp	x	x	float_2_gp_ex6
testbit	x	x	testbit_ex6
byte_boolean	x	x	byte_2_bool_ex6
boolean_byte	x	x	bool_2_byte_ex6
kf_integer	x	x	kf_2_integer_ex6
swab_buffer *[2]	x	x	swab_buffer_ex6
copy_buffer *[2]	x	x	copy_buffer_ex6
USHORT_2_bcd *[2]	x	x	ushort_2_bcd_ex6
bcd_2_USHORT *[2]	x	x	bcd_2_ushort_ex6
Teleservice Functions			
ts_dial	x		ts_dial_ex6
ts_hang_up_dial	x		ts_hang_up_dial_ex6
ts_set_ringindicator	x		ts_set_ringindicator_ex6
ts_read_info	x		ts_read_info_ex6
ts_hang_up_ring	x		ts_hang_up_ring_ex6
ts_get_modem_name *[2]	x		ts_get_modem_name_ex6

*[2] These functions are not described in the PRODAVE S7 V5.6 documentation !

4.1 Basic Functions

4.1.1 load_tool

This function initializes the adapter, checks if the driver is loaded, initializes the configured addresses and switches the selected interface to active.

With **load_tool** the transport connection is established via MPI/PB addresses.

int load_tool (char *chConNo*, char* *pstrAccessPoint,* char* *pConTable*);

Parameters

chConNo

[in] Number of the connection

pstrAccessPoint

[in] Access point (zero-terminated) of the used driver, e.g. "S7ONLINE" for the MPI driver or 0 (default).

pConTable

[in] pointer to address list of connected users. ‘cAdr ==0’ is taken as the end mark of the list.

Return Values

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**).

Call Example

A maximum of 32 connections can be established using the AS300/400.

Only one connection can be established using the AS200.

Structure of the address lists:

#pragma pack(1)	
typedef struct {	
unsigned char *cAdr;*	// station address
unsigned char *cSegmentId;*	// segment ID
unsigned char *cSlotNo;*	// slot number
unsigned char *cRackNo;*	// rack number
} adr_table_type;	
#pragma pack(1)	

Each user is specified with the entry in the address list:

cAdr	Station address of the user, default: 2
cSegmentId	Segment ID of the user, default: 0 (reserved for later expansions)
cSlotNo	Slot number of the user, default: 1
cRackNo	Rack number of the user, default: 0

Requirements

	V5.6 load_tool, load_tool_ex	**V6.0 LoadConnection_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

unload_tool, new_ss, LoadConnection_ex6

4.1.2 unload_tool

This function deinitializes the connections and the adapter and must be called prior to leaving the application.

unload_tool deinitializes connections which were initialized with **load_tool**.

int unload_tool (void);

Parameters

None

Return Values

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**).

Call Example

None

Requirements

	V5.6 unload_tool	**V6.0 UnloadConnection_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

load_tool, **new_ss, UnloadConnection_ex6**

4.1.3 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.

int new_ss (char *chConNo***);**

Parameters

chConNo

[in] Number of the connection to be activated.

Return Values

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**).

Call Example

None

Requirements

	V5.6 new_ss	**V6.0 SetActiveConnection_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

load_tool, **unload_tool, SetActiveConnection_ex6**

4.2 Functions for Data Communication Traffic to the S7 300/400

4.2.1 ag_info

The function **ag_info** reads the PLC software version and the PG interface version as well as the MLFB number of the PLC and stores them in a transfer buffer of the PG/PC as an ASCII string zero-terminated.

int ag_info (void * *Buffer*);

Parameters

Buffer

[out] Transfer buffer with the PLC information to be supplied

Return Values

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**).

Call Example

char	Buffer[MAX_BUFFER];
unsigned short * *wBuffer* = (unsigned short *)*Buffer*;	// Word by word access

The versions must be interpreted as integer values, the MLFB numbers as ASCII values.

wBuffer[0]	Integer value PLC version
wBuffer[2]	Integer value PGAS version
Buffer[4] ...Buffer[24]	ASCII value MLFB of the connected PLC

Requirements

	V5.6 ag_info	**V6.0 as_info_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

ag_zustand, as_info_ex6

4.2.2 ag_zustand

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

int ag_zustand (void * *Buffer*);

Parameters

Buffer

[out] Transfer buffer with the PLC status to be supplied

Return Values

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**).

Call Example

```
char        Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // Byte-by-byte unsigned
                                                         access

cBuffer[0]          == 0          PLC is in RUN mode
cBuffer[0]          != 0          PLC is in STOP or in STARTUP mode
```

Requirements

	V5.6 ag_zustand	**V6.0 as_zustand_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

ag_info, as_zustand_ex6

4.2.3 db_buch

The function **db_buch** checks which DBs exist in the PLC. For this purpose a transfer buffer of 512 words must be made available, for each block one word. If the value in the referenced buffer is = 0 this means that the assigned block DB does not exist.

int db_buch (void * *Buffer*);

Parameters

Buffer

[out] transfer buffer with the DB list to be supplied

Return Values

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**).

Call Example

```
char        Buffer[MAX_BUFFER];
unsigned short * wBuffer = (unsigned short *)Buffer;        // Word by word access

wBuffer[0]      != 0        DB0 exists
wBuffer[24]     == 0        DB24 does not exist
wBuffer[511]    != 0        DB511 exists
```

Requirements

	V5.6 db_buch	V6.0 db_buch_ex6
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

db_buch_ex6

4.2.4 db_read

The function **db_read** reads an amount of data words from a data block in the PLC and transfers them into a transfer buffer of the PG/PC.

With **db_read** it is possible to read out data words word-by-word (pnAnzahl = 1) or block-by-block (pnAnzahl > 1)**.**

int db_read (int *nBstNo*, int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nBstNo

[in] Number of the data block

nStartNo

[in] Start number of the first data word to be read

pnAnzahl

[in/out] Amount of data words to be read

Buffer

[out] Transfer buffer for the data words which were read

Return Values

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**).

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 contents of *pnAnzahl* is corrected and error message 303 hex is returned.

Call Example

```
char          Buffer[MAX_BUFFER];

unsigned short * wBuffer = (unsigned short *)Buffer;          // Word by word
access
```

Attention: 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. The functions **kf_integer** and **swab_buffer** can be used to swap bytes.

Requirements

	V5.6 db_read	**V6.0 db_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

db_write, kf_integer, swab_buffer, db_read_ex6

4.2.5 d_field_read

The function **d_field_read** reads an amount of data bytes from a data block in the PLC and transfers them in the transfer buffer of the PG/PC.

int d_field_read (int *nBstNo*, int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nBstNo

[in] Number of the data block

nStartNo

[in] Start number of the first data byte to be read

pnAnzahl

[in/out] Amount of data bytes to be read

Buffer

[out] Transfer buffer for the data bytes which were read

Return Values

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**).

Call Example

```
char          Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Requirements

	V5.6 d_field_read	**V6.0 field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

e|a|m_field_read, t|z_field_read, field_read_ex6

4.2.6 e|a|m_field_read

These functions read an amount of bytes from the PLC and transfer them to a transfer buffer of the PG/PC.

Wtih **e_field_read** input bytes can be read.

With **a_field_read** output bytes can be read.

With **m_field_read** flag bytes can be read.

int e_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int a_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int m_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first byte to be read

pnAnzahl

[in/out] Amount of bytes to be read

Buffer

[out] Transfer buffer for the bytes which were read

Return Values

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**).

Call Example

```
char          Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Requirements

	V5.6 e_field_read, …	**V6.0 field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

d_field_read, t|z_field_read, field_read_ex6

4.2.7 t|z_field_read

These functions read an amount of words from the PLC and transfer them to a transfer buffer of the PG/PC.

With **t_field_read** timer words can be read.

With **z_field_read** counter words can be read.

int t_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int z_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first word to be read

pnAnzahl

[in/out] Amount of words to be read

Buffer

[out] Transfer buffer for the words which were read

Return Values

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**).

Call Example

```
char        Buffer[MAX_BUFFER];
unsigned short * wBuffer = (unsigned short *)Buffer;        // Word by word access
```

Requirements

	V5.6 t_field_read	**V6.0 field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

d|e|a|m_field_read, field_read_ex6

4.2.8 mix_read

The function **mix_read** reads the data configured with "data" from a data block in the PLC and transfers them to a transfer buffer of the PG/PC.

With **mix_read,** "Size" can be a byte or a word.

int mix_read (char * *pData*, void * *Buffer*);

Parameters

pData

[in] Pointer on a type list. Type element contents == 0 is taken as the end mark of the list.

Buffer

[out] Transfer buffer for the data bytes/words/ double words which were read

Return Values

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**).

Call Example

This function enables the user to read mixed data. A maximum of 20 list items can be created. The type list can be accessed via a structure:

```
#pragma pack(1)
    typedef struct {
    unsigned char Typ;
    unsigned char Size;
    unsigned short nBstNo;
    unsigned short nDatNo;
    } mix_tab_type;
#pragma pack(1)
```

Typ

The following data can be read (small or large ASCII characters)

e	input bytes
a	output bytes
m	flag bytes
t	timer words
z	counter words
d	data from DB

Size

The data to be read can have the following data types (small or large ASCII characters):

b,w	byte or word	for the input bytes
b,w	byte or word	for the output bytes
b,w	byte or word	for the flag bytes
w	word	for the timer words
w	word	for the counter words

nBstNo

Number of the data block

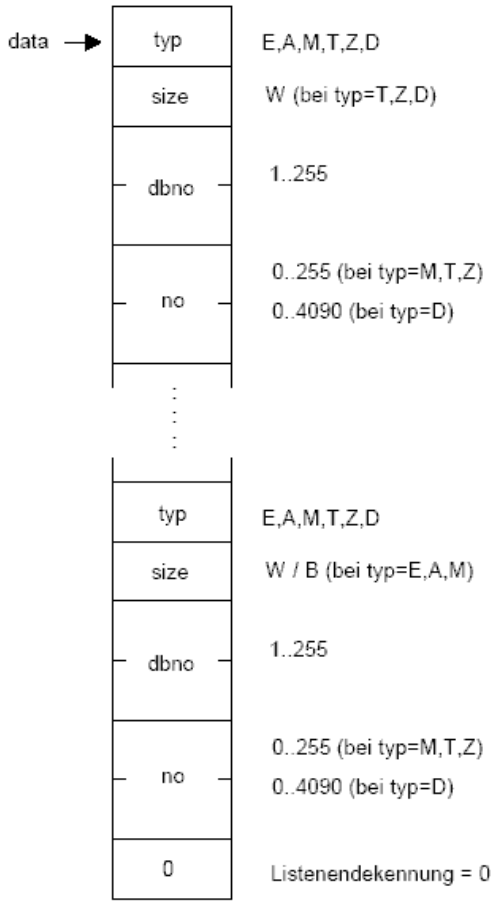
nDatNo

Number of the data byte/word/double word to be read

For *Typ* m, t and z 0 ... 255

For *Typ* d 0 ... 4090

"data" must have the following structure:



Size = 'b' read byte and enter it into the buffer

Size = 'w' read word and enter it into the buffer

The read values are entered in sequence in the buffer. I.e. the user himself must carry out structured processing of the field occupied with the read values:

```
char        Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // Byte-by-byte unsigned
                                                          access
unsigned short * wBuffer = (unsigned short *)Buffer; // Word-by-word access
```

Attention: 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. The functions **kf_integer** and **swab_buffer** can be used to swap bytes.

Requirements

	V5.6 mix_read	**V6.0 field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

mix_write, kf_integer, **swab_buffer, field_read_ex6**

4.2.9 db_write

The function **db_write** writes an amount of data words from a transfer buffer of the PG/PC into a data block of the PLC.

It is possible to read out data words word-by-word (pnAnzahl = 1) or block-by-block (pnAnzahl > 1) with **db_write**.

int db_ write (int *nBstNo*, int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nBstNo

[in] Number of the data block

nStartNo

[in] Start number of the first data word to be written

pnAnzahl

[in/out] Amount of data words to be written

Buffer

[in] Transfer buffer for the data words to be written

Return Values

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**).

If the data block does not exist, this is indicated by a return value = error number.

Call Example

```
char          Buffer[MAX_BUFFER];
unsigned short * wBuffer = (unsigned short *)Buffer; // Word by word access
```

Attention: The data words must be stored in the "buffer" not in accordance with Intel notation (low byte - high byte) but in STEP5 notation (high byte - low byte) when writing to a data block. The functions **kf_integer** and **swab_buffer** can be used to swap bytes.

Requirements

	V5.6 db_write	**V6.0 db_write_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

db_read, kf_integer, **swab_buffer, db_write_ex6**

4.2.10 d_field_write

The function **d_field_write** writes an amount of data bytes from a transfer buffer of the PG/PC into a data block of the PLC.

int d_field_write (int *nBstNo*, int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nBstNo

[in] Number of the data block

nStartNo

[in] Start number of the first data byte to be written

pnAnzahl

[in/out] Amount of data bytes to be written

Buffer

[in] Transfer buffer for the data bytes to be written

Return Values

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**).

Call Example

```
char          Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Requirements

	V5.6 d_field_write	**V6.0 field_write_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

e|a|m_field_write, z_field_write, field_write_ex6

4.2.11 a|m_field_write

The functions write an amount of bytes from a transfer buffer of the PG/PC into the PLC.

With **a_field_write** output bytes can be written.

With **m_field_write** flag bytes can be written.

int a_field_ write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int m_field_ write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first byte to be written

pnAnzahl

[in/out] Amount of bytes to be written

Buffer

[out] Transfer buffer for the bytes which were written

Return Values

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**).

Call Example

```
char        Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Requirements

	V5.6 a_field_write, ...	**V6.0 field_write_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

d_field_write, z_field_write, field_write_ex6

4.2.12 z_field_write

The function **z_field_write** writes an amount of counter words from a transfer buffer of the PG/PC into the PLC.

int z_field_ write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first counter word to be written

pnAnzahl

[in/out] Amount of counter words to be written

Buffer

[out] Transfer buffer for the counter words to be written

Return Values

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**).

Call Example

```
char        Buffer[MAX_BUFFER];
unsigned short * wBuffer = (unsigned short *)Buffer; // Word-by-word access
```

Requirements

	V5.6 z_field_write	V6.0 field_write_ex6
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

d|a|m_field_write, field_write_ex6

4.2.13 mix_write

The function **mix_write** writes the data configured with "data" from a transfer buffer of the PG/PC into a data block of the PLC.

With **mix_write,** "Size" can be a byte or a word.

int mix_ write (char * *pData*, void * *Buffer*);

Parameters

pData

[in] Pointer on a type list. Type element contents == 0 is taken as the end mark of the list.

Buffer

[out] Transfer buffer for the data bytes/words/ double words to be written

Return Values

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**).

Call Example

This function enables the user to write mixed data. A maximum of 20 list entries can be created. The type list can be accessed via a structure:

```
#pragma pack(1)
typedef struct {
unsigned char Typ;
unsigned char Size;
unsigned short nBstNo;
unsigned short nDatNo;
    } mix_tab_type;
#pragma pack(1)
```

Typ

The following data can be written (small or large ASCII characters):

e/E	input bytes
a/A	output bytes
m/M	flag bytes
t/T	timer words
z/Z	counter words
d/D	Daten in DB

Size

The data to be written can have the following data types (small or large ASCII characters):

b,w	byte or word	for the input bytes
b,w	byte or word	for the output bytes
b,w	byte or word	for the flag bytes
w	word	for the timer words
w	word	for the counter words

nBstNo

Number of the data block

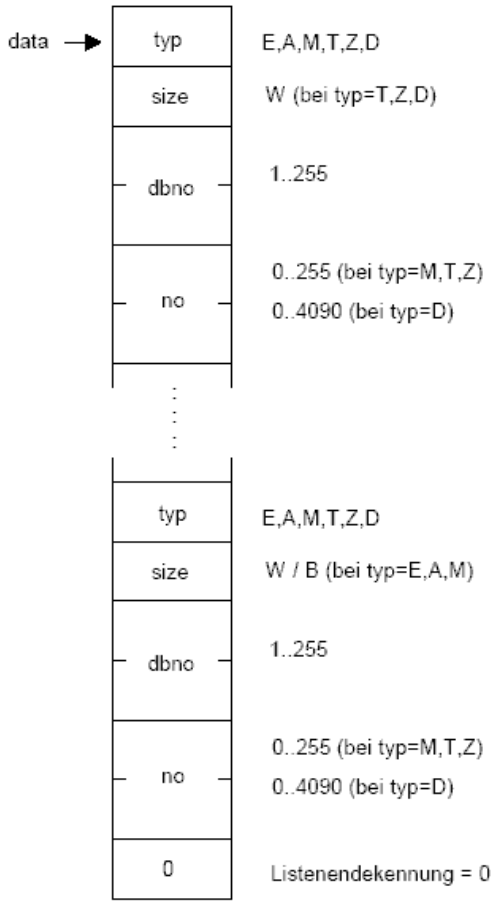
nDatNo

Number of the data byte/word/double word to be written

For *Typ* m, t and z 0 ... 255

For *Typ* d 0 ... 4090

"data" must have the following structure:



Size = 'b' Enter byte in buffer

Size = 'w' Enter word in buffer

The values to be written have to be entered in sequence in the buffer:

```
char        Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // Byte-by-byte unsigned
                                                         access
unsigned short * wBuffer = (unsigned short *)Buffer; // Word-by-word access
unsigned long * dwBuffer = (unsigned long *)Buffer; // Double word-by-word
access
```

Attention: The data words must be stored in the "buffer" not in accordance with Intel notation (low byte - high byte) but in STEP5 notation (high byte - low byte) when writing to a data block. The functions **kf_integer** and **swab_buffer** can be used to swap bytes.

Requirements

	V5.6 mix_write	**V6.0 field_write_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

mix_read, kf_integer, **swab_buffer, field_write_ex6**

4.2.14 mb_setbit

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

int mb_setbit (int n*MbNo*, int n*BitNo*);

Parameters

nMbNo

[in] number of the flag byte

nBitNo

[in] bit number in the flag byte

Return Values

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**).

Call Example

None

Requirements

	V5.6 mb_setbit	**V6.0 mb_setbit_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

mb_resetbit, **mb_bittest, mb_setbit_ex6**

4.2.15 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.

int mb_resetbit (int n*MbNo*, int n*BitNo*);

Parameters

nMbNo

[in] number of the flag byte

nBitNo

[in] bit number in the flag byte

Return Values

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**).

Call Example

None

Requirements

	V5.6 mb_resetbit	**V6.0 mb_setbit_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

mb_setbit, **mb_bittest, mb_setbit_ex6**

4.2.16 mb_bittest

The function **mb_bittest** checks a bit in a specified flag byte and supplies the status of the specified bit in *bitwert.

int mb_bittest (int n*MbNo*, int n*BitNo,* char * bitwert);

Parameters

nMbNo

[in] Number of the flag byte

nBitNo

[in] Bit number in the flag byte

bitwert

[out] Transfer buffer with the tested bit value

Return Values

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**).

Call Example

None

Requirements

	V5.6 mb_bittest	**V6.0 mb_bittest_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

mb_setbit, **mb_resetbit, mb_bittest_ex6**

4.3 Functions for Data Communication Traffic to the S7 200

4.3.1 as200_ag_info

The function **as200_ag_info** reads the PLC software version and the PG interface version as well as the PLC type of the PLC and stores them in a transfer buffer of the PG/PC as an ASCII string zero-terminated.

int as200_ag_info (void * *Buffer*);

Parameters

Buffer

[out] Transfer buffer with the PLC information to be supplied

Return Values

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**).

Call Example

char Buffer[MAX_BUFFER];

unsigned short * *wBuffer* = (unsigned short *)*Buffer*; // Word by word access

The versions must be interpreted as integer values, the MLFB numbers as ASCII values.

wBuffer[0]	Integer value PLC version
wBuffer[2]	Integer value PGAS version
Buffer[4] ...Buffer[24]	ASCII value PLC type of the connected PLC

Requirements

	V5.6 as200_ag_info	**V6.0 as200_as_info_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_ag_zustand, as200_as_info_ex6

4.3.2 as200_ag_zustand

The function **as200_ag_zustand** reads the PLC status (RUN or STOP) from the PLC and stores the data in a storage area of the PG/PC.

int as200_ag_zustand (void * *Buffer*);

Parameters

Buffer

[out] Transfer buffer with the PLC status to be supplied

Return Values

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**).

Call Example

```
char        Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

cBuffer[0]	== 0	PLC is in RUN mode
cBuffer[0]	!= 0	PLC is in STOP or in STARTUP mode

Requirements

	V5.6 as200_ag_zustand	**V6.0 as200_as_zustand_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_ag_info, as200_as_zustand_ex6

4.3.3 as200_e|a|m|sm|vs_field_read

These functions read an amount of bytes from the PLC and transfer them to a transfer buffer of the PG/PC.

With **as200_e_field_read** input bytes can be read.

With **as200_a_field_read** output bytes can be read.

With **as200_m_field_read** flag bytes can be read.

With **as200_sm_field_read** special flag bytes can be read.

With **as200_vs_field_read** variable memory bytes can be read.

int as200_e_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_a_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_m_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_sm_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_vs_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first byte to be read

pnAnzahl

[in/out] Amount of bytes to be read

Buffer

[out] Transfer buffer for the bytes which were read

Return Values

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**).

Call Example

```
char          Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Requirements

	V5.6 as200_e_field_read, ...	**V6.0 as200_field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_t|z_field_read, as200_mix_read, as200_field_read_ex6

4.3.4 as200_t_field_read

The function **as200_t_field_read** reads an amount of timer values from PLC and transfers them to a transfer buffer of the PG/PC.

int as200_t_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first timer value to be read

pnAnzahl

[in/out] Amount of timer values to be read

Buffer

[out] Transfer buffer for the timer values which were read

Return Values

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**).

Call Example

5 bytes are received per timer value, and only 2 bytes contain the requested value.

In the following example the timer values 0 and 1 are read:

char	Buffer[MAX_BUFFER];
unsigned short T0, T1;	// Timer word
int error;	
error = as200_t_field_read((int)0, (int)2, Buffer);	
T0 = (unsigned short)Buffer[4] | (unsigned short)Buffer[3] << 8;	
T1 = (unsigned short)Buffer[9] | (unsigned short)Buffer[8] << 8;	

Requirements

	V5.6 as200_t_field_read	**V6.0 as200_field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_e|a|m|sm|vs_field_read, as200_z_field_read, as200_mix_read, as200_field_read_ex6

4.3.5 as200_z_field_read

The function **as200_z_field_read** reads an amount of counter values from PLC and transfers them to a transfer buffer of the PG/PC.

int z_field_read (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first counter value to be read

pnAnzahl

[in/out] Amount of counter values to be read

Buffer

[out] Transfer buffer for the counter values which were read

Return Values

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**).

Call Example

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

In the following example the counter values 3 and 4 are read:

char	Buffer[MAX_BUFFER];
unsigned short Z3, Z4;	// counter word
int error;	
error = as200_z_field_read((int)3, (int)2, Buffer);	
Z3 = (unsigned short)Buffer[2] | (unsigned short)Buffer[1] << 8;	
Z4 = (unsigned short)Buffer[5] | (unsigned short)Buffer[4] << 8;	

Requirements

	V5.6 as200_z_field_read	**V6.0 as200_field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_e|a|m|sm|vs_field_read, as200_t_field_read, as200_mix_read, as200_field_read_ex6

4.3.6 as200_mix_read

The function **as200_mix_read** reads the data configured with "data" from a data block in the PLC and transfers them to a transfer buffer of the PG/PC.

int as200_mix_read (char * *pData*, void * *Buffer*);

Parameters

pData

[in] Pointer on a type list. Type element contents == 0 is taken as the end mark of the list.

Buffer

[out] Transfer buffer for the data bytes which were read

Return Values

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**).

Call Example

This function enables the user to read mixed data. A maximum of 20 list entries can be created. The type list can be accessed via a structure:

```
#pragma pack(1)
typedef struct {
unsigned char Typ;
unsigned char Size;
unsigned short nBstNo;
unsigned short nDatNo;
    } mix_tab_type;
#pragma pack(1)
```

Typ

The following data can be read (small or large ASCII characters)

e	input bytes
a	output bytes
m	flag bytes
v	variable memory words
s	special flag bytes

Size

For all data to be read the data type must be byte (small or large ASCII characters):

b Byte for all data

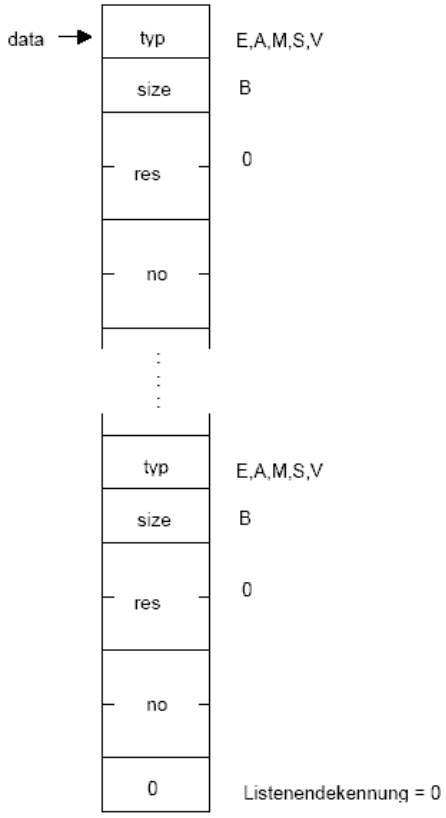
nBstNo

Number of the data block

nDatNo

Number of the data byte to be read

"data" must have the following structure:



Size = 'b' read byte and enter it into the buffer

The read values are entered in sequence in the buffer. I.e. the user himself must carry out structured processing of the field occupied with the read values:

```
char          Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Attention: 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. The functions **kf_integer** and **swab_buffer** can be used to swap bytes.

Requirements

	V5.6 as200_mix_read	**V6.0 as200_field_read_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_mix_write, kf_integer, **swab_buffer, as200_field_read_ex6**

4.3.7 as200_a|m|sm|vs_field_write

The functions write an amount of bytes from a transfer buffer of the PG/PC into the PLC.

With **as200_a_field_write** output bytes can be written.

With **as200_m_field_write** flag bytes can be written.

With **as200_sm_field_write** special flag bytes can be written.

With **as200_vs_field_write** variable memory bytes can be written.

int as200_a_field_write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_m_field_write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_sm_field_write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

int as200_vs_field_write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first byte to be written

pnAnzahl

[in/out] Amount of bytes to be written

Buffer

[out] Transfer buffer for the bytes which were written

Return Values

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**).

Call Example

```
char          Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Requirements

	V5.6 as200_e_field_write, ...	**V6.0 as200_field_write_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_z_field_write, as200_mix_write, as200_field_write_ex6

4.3.8 as200_z_field_write

The function **as200_z_field_write** writes an amount of counter values from a transfer buffer of the PG/PC into the PLC.

int as200_z_field_write (int *nStartNo*, int * *pnAnzahl*, void * *Buffer*);

Parameters

nStartNo

[in] Start number of the first counter value to be written

pnAnzahl

[in/out] Amount of counter values to be written

Buffer

[out] Transfer buffer for the counter values to be written

Return Values

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**).

Call Example

3 bytes are sent per counter value, and only 2 bytes contain the specified value.

Requirements

	V5.6 as200_z_field_write	**V6.0 as200_field_write**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_a|m|sm|vs_field_write, as200_mix_write, as200_field_write_ex6

4.3.9 as200_mix_write

The function **as200_mix_write** writes the data configured with "data" from a transfer buffer of the PG/PC into a data block in the PLC.

int as200_mix_write (char * *pData*, void * *Buffer*);

Parameters

pData

[in] Pointer on a type list. Type element contents == 0 is taken as the end mark of the list.

Buffer

[out] Transfer buffer for the data bytes which were written

Return Values

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**).

Call Example

This function enables the user to write mixed data. A maximum of 20 list entries can be created. The type list can be accessed via a structure:

```
#pragma pack(1)
typedef struct {
unsigned char Typ;
unsigned char Size;
unsigned short nBstNo;
unsigned short nDatNo;
    } mix_tab_type;
#pragma pack(1)
```

Typ

The following data can be written (small or large ASCII characters):

e	input bytes
a	output bytes
m	flag bytes
v	variable memory words
s	special flag bytes

Size

For all data to be written the data type must be byte (small or large ASCII characters):

b Byte for all data

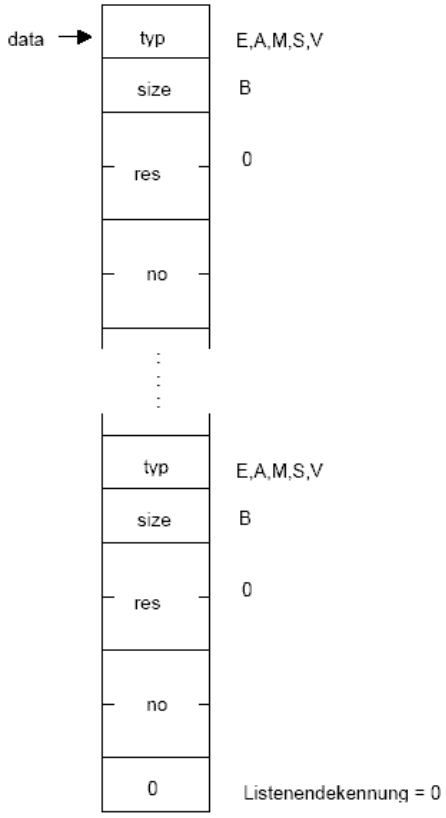
nBstNo

Number of the data block

nDatNo

Number of the data byte to be written

"data" must have the following structure:



Size = 'b' write byte

The values to be written have to be entered in sequence into the buffer.

```
char        Buffer[MAX_BUFFER];
unsigned char * cBuffer = (unsigned char *)Buffer;    // byte-by-byte unsigned
                                                         access
```

Attention: 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. The functions **kf_integer** and **swab_buffer** can be used to swap bytes.

Requirements

	V5.6 as200_mix_write	**V6.0 as200_field_write_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_mix_read, kf_integer, **swab_buffer, as200_field_write_ex6**

4.3.10 as200_mb_setbit

Fhe function **as200_mb_setbit** sets a flag in the PLC to 1. It is not checked whether the flag bit exists in the used PLC.

int as200_mb_setbit (int n*MbNo*, int n*BitNo*);

Parameters

nMbNo

[in] number of the flag byte

nBitNo

[in] bit number in the flag byte

Return Values

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**).

Call Example

None

Requirements

	V5.6 as200_mb_setbit	**V6.0 as200_mb_setbit_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_mb_resetbit, **as200_mb_bittest, as200_mb_setbit_ex6**

4.3.11 as200_mb_resetbit

The function **as200_mb_resetbit** sets a flag in the PLC to 0. It is not checked whether the flag bit exists in the used PLC.

int as200_mb_resetbit (int n*MbNo*, int n*BitNo*);

Parameters

nMbNo

[in] number of the flag byte

nBitNo

[in] bit number in the flag byte

Return Values

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**).

Call Example

None

Requirements

	V5.6 as200_mb_resetbit	**V6.0 as200_mb_setbit_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_mb_setbit, **as200_mb_bittest, as200_mb_setbit_ex6**

4.3.12 as200_mb_bittest

The function **as200_mb_bittest** checks a bit in a specified flag byte and supplies the status of the specified bit in *bitwert.

int as200_mb_bittest (int n*MbNo*, int n*BitNo,* char * bitwert);

Parameters

nMbNo

[in] Number of the flag byte

nBitNo

[in] Bit number in the flag byte

bitwert

[out] Transfer buffer with the tested bit value

Return Values

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**).

Call Example

None

Requirements

	V5.6 as200_mb_bittest	**V6.0 as200_mb_bittest_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

as200_mb_setbit, **as200_mb_resetbit, as200_mb_bittest_ex6**

4.4 Comfort Functions

4.4.1 error_message

The function **error_message** supplies the approprioate error text relating to an error message in the form of a zero terminated character string.

int error_message (int *nErrorNnr*, char * *Buffer*);

Parameters

nErrorNnr

[in] Error number

Buffer

[out] Transfer buffer with the error texts

Return Values

If there were no errors, the function supplies 0 as the return value. If an error occurred, the function supplies the following values:

- ERROR.DAT file does not exist or cannot be opened
- Error when reading the ERROR.DAT file
- Wrong structure of the ERROR.DAT file
- No error text exists for this error number
- Too many error texts in ERROR.DAT

Call Example

A maximum of 100 error texts can be stored in the ERROR.DAT file.

When transferring error number 0, the file name of the error text file to be loaded can be transferred in "buffer" (for example, an English or German error text file). 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 located in the same directory as the program.

The ERROR.DAT file is read when the function is first called and the texts are stored in a field.

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 constant processing time for further calls of this function.

Structure of the error text file

[Error number as ASCII hex]: [error text]

You can find the error texts in the appendix.

Requirements

	V5.6 error_message	**V6.0 GetErrorMessage_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

GetErrorMessage_ex6

4.4.2 kg_to_float

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

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

Parameters

kg

[in] S5 floating point value

ieee

[out] Float value

Return Values

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

Call Example

None

Requirements

	V5.6 kg_to_float	**V6.0 kg_2_float_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

float_to_kg, kg_2_float_ex6

4.4.3 float_to_kg

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

int float_to_kg (void * *ieee,* void * *kg*);

Parameters

ieee

[in] Float value

kg

[out] S5 floating point value

Return Values

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

Call Example

None

Requirements

	V5.6 float_to_kg	**V6.0 float_2_kg_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

kg_to_float, float_2_kg_ex6

4.4.4 gp_to_float

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

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

Parameters

gp

[in] S7 floating point value

ieee

[out] Float value

Return Values

None

Call Example

None

Requirements

	V5.6 gp_to_float	V6.0 gp_2_float_ex6
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

float_to_gp, gp_2_float_ex6

4.4.5 float_to_gp

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

void float_to_gp (void * *ieee,* void * *gp*);

Parameters

ieee

[in] Float value

gp

[out] S7 floating point value

Return Values

None

Call Example

None

Requirements

	V5.6 float_to_gp	V6.0 float_2_gp_ex6
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

gp_to_float, float_2_gp_ex6

4.4.6 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.

char testbit (char *Wert,* char *BitNo*);

Parameters

Wert

[in] Value of the byte variable

BitNo

[in] Bit to be tested in the byte variable

Return Values

Return value TRUE (or 1):	Bit is set (or 1)
Return value FALSE (or 1):	Bit is not set (or 0)

Call Example

None

Requirements

	V5.6 testbit	**V6.0 testbit_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

See also:

testbit_ex6

4.4.7 byte_boolean

The function **byte_boolean** converts a byte to eight logical values (PC display).

void byte_boolean (char *Wert*, char * *Buffer*);

Parameters

Wert

[in] Byte value

Buffer

[out] Pointer on buffer with eight converted logical values

Return Values

None

Call Example

The transferred pointer should point to a char field with the following structure:

Buffer[0]	*Buffer*[1]	*Buffer*[2]	*Buffer*[3]	*Buffer*[4]	*Buffer*[5]	*Buffer*[6]	*Buffer*[7]
bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

Requirements

	V5.6 byte_boolean	**V6.0 byte_2_bool_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

boolean_byte, byte_2_bool_ex6

4.4.8 boolean_byte

The function **boolean_byte** converts eight logical values (PC display) to a byte.

char boolean_byte (char * *Buffer*);

Parameters

Buffer

[in] Pointer on buffer with eight logical values

Return Values

Converted byte value

Call Example

The transferred pointer should point to a char field with the following structure:

Buffer[0]	*Buffer*[1]	*Buffer*[2]	*Buffer*[3]	*Buffer*[4]	*Buffer*[5]	*Buffer*[6]	*Buffer*[7]
bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

Requirements

	V5.6 boolean_byte	**V6.0 bool_2_byte_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

byte_boolean, bool_2_byte_ex6

4.4.9 kf_integer

The function swaps the high byte and the low byte of a transferred value.

With **kf_integer** the high and low bytes of a 16-bit value are swapped.

unsigned short kf_integer (unsigned short w*Wert*);

Parameters

wWert

[in] 16bit values

Return Values

kf_integer returns a 16-bit value with swapped bytes.

Call Example

None

Requirements

	V5.6 kf_integer	**V6.0 kf_2_integer_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

swab_buffer, kf_2_integer_ex6

4.4.10 swab_buffer

The function **swab_buffer** swaps the high byte and the low byte of a transferred buffer.

void swab_buffer (void * *Buffer*, int *nAnzahl*);

Parameters

Buffer

[in/out] Pointer on buffer in which the bytes are swapped

nAnzahl

[in] Amount of bytes to be swapped

Return Values

None

Call Example

Internally, the Standard C function void _swab(char * src, char * dest, int n) is called.

Requirements

	V5.6 swab_buffer	**V6.0 swab_buffer_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

kf_integer, copy_buffer, swab_buffer_ex6

4.4.11 copy_buffer

The function **copy_buffer** copies an amount of bytes from one buffer to the other.

void copy_buffer (void * *ZielBuffer*, void * *QuellBuffer*, int *nAnzahl*);

Parameters

ZielBuffer

[in/out] Pointer on buffer in which the bytes are copied

QuellBuffer

[in] Pointer on buffer, from which the bytes are taken

nAnzahl

[in] Amount of bytes to be copied

Return Values

None

Call Example

Internally, the Standard C function void *memcpy(char * dest, char * src, size_t count) is called.

Requirements

	V5.6 copy_buffer	**V6.0 copy_buffer_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

swab_buffer, copy_buffer_ex6

4.4.12 USHORT_2_bcd

The functions are conversion routines that convert a number of binary values into BCD values.

USHORT_2_bcd converts 16-bit values (words).

void USHORT_2_bcd (unsigned short * *pwWerte*, unsigned short *wAnzahl*, char *InBytechange*, char *OutBytechange*);

Parameters

pwWerte

[in/out] Pointer on 16-bit dual values

wAnzahl

[in] Amount of values

InBytechange

[in] Boolean expression TRUE or FALSE

OutBytechange

[in] Boolean expression TRUE or FALSE

Return Values

If the parameter 'InBytechange' is set (1) then the high and low bytes will be swapped before the conversion to BCD values. If however, the parameter 'OutBytechange' is set, the high and low bytes will be swapped after the conversion.

If neither of the ByteChange arguments is set, then no high-low conversion will take place.

After calling the function, *pwWerte* points to 16-bit BCD values.

Call Example

With this function you can, for example, set counters or provide time functions.

The available value range for 16-bit BCD values is +999 to -999.

Requirements

	V5.6 USHORT_2_bcd	**V6.0 ushort_2_bcd_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

bcd_2_USHORT, ushort_2_bcd_ex6

4.4.13 bcd_2_USHORT

The functions are conversion routines that convert a number of BCD values into binary values.

bcd_2_ushort_ex6 converts 16-bit values (words).

void bcd_2_ USHORT (unsigned short * *pwWerte*, unsigned short *wAnzahl*, char *InBytechange*, char *OutBytechange*);

Parameters

pwWerte

[in/out] Pointer on 16-bit BCD values

wAnzahl

[in] Amount of values

InBytechange

[in] Boolean expression TRUE or FALSE

OutBytechange

[in] Boolean expression TRUE or FALSE

Return Values

After calling the function, *pwWerte points to* 16-bit dual values.

Call Example

With this function you can, for example, set counters or provide time functions.

The available value range for 16-bit BCD values is +999 to -999.

Requirements

	V5.6 bcd_2_USHORT	**V6.0 bcd_2_ushort_ex6**
Windows:	95, ME, NT4, 2000, XP	
Header:	KOMFORT.H	
Library:	KOMFORT.DLL	

See also:

USHORT_2_bcd, bcd_2_ushort_ex6

4.5 Teleservice Functions

4.5.1 ts_dial

The function **ts_dial** dials a remote station via the modem and establishes the connection to the TS-Adapter.

int ts_dial (char * *cModemName*, char * *cStandort*, char * *cTelNo*, char * *cUserName*, char * *cPassword*, HANDLE *WindowHandle*, unsigned int *Message*, WPARAM *wParam*, char * *Res1*);

Parameters

cModemName

[in] Name of the modem to be used, can be adjusted in Control Panel / Modems / Dialing Rules.

cStandort

[in] Name of the modem location, can be adjusted in Control Panel / Modems / Dialing Rules.

cTelNo

[in] Telephone number, which is dialled by the connected modem.

cUserName

[in] Here you specify the user name configured in the TS-Adapter to be called.

cPassword

[in] Here you specify the password configured in the TS-Adapter to be called.

WindowHandle

[in] Here you can transfer a window handle.

Message

[in] Message which is sent to the window when the connection has been established or the timeout has elapsed.

wParam

[in] Parameters for the message.

Res1

[in] Reserved for later entries, must be set to NULL now.

Return Values

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**).

Note

When function **ts_dial** is called asynchronously, no error number will be generated in the event of error. In this case, inform the user and allow him/her to decide whether the function should be called again or if more detailed error information is required. For detailed information, please use synchronous call.

Call Example

None

Requirements

	V5.6 ts_dial	
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

4.5.2 ts_hang_up_dial

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

int ts_hang_up_dial (void);

Parameters

None

Return Values

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**).

Call Example

None

Requirements

	V5.6 ts_hang_up_dial	
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

4.5.3 ts_set_ringindicator

The function **ts_set_ringindicator** initializes the sub-ordinate system for receiving calls, establishing the connection and reporting (ring indication).

int ts_set_ringindicator (char * *cModemName*, char * *cNumberOfRings*, HANDLE *WindowHandle*, unsigned int *Message*, WPARAM *wParam*, char * *Res1*);

Parameters

cModemName

[in] Name of the modem to be used for the ring indication, can be selected in Control Panel / Modem.

cNumberOfRings

[in] Number of rings until the modem replies.

WindowHandle

[in] Here you can transfer a window handle.

Message

[in] Message which is sent to the window when the connection has been established or the timeout has elapsed.

wParam

[in] Parameters for the message.

Res1

[in] Reserved for later entries, must be set to NULL now.

Return Values

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**).

Call Example

None

Requirements

	V5.6 ts_set_ringindicator	
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

4.5.4 ts_read_info

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

int ts_read_info (void * *IventId*, unsigned char * *MpiAdr*);

Parameters

IventId

[in] Pointer to a field 16 bytes long. Here you can enter information on the alarm triggering station.

MpiAdr

[in] Pointer to a byte. Here you can enter the MPI address of the alarm triggering station.

Return Values

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**).

Call Example

None

Requirements

	V5.6 ts_read_info	
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

4.5.5 ts_hang_up_ring

The function **ts_hang_up_ring** interrupts the connection established by the TS-Adapter.

int ts_hang_up_ring (void);

Parameters

None

Return Values

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**).

Call Example

None

Requirements

	V5.6 ts_hang_up_ring	
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

4.5.6 ts_get_modem_name

The function **ts_get_modem_name** supplies all modem names known to the system.

int ts_get_modem_name (int *ModemId*, char * *Buffer*, int * *BufferLen*);

Parameters

ModemId

[in] Modem ID 0 ... n

Buffer

[out] Pointer to buffer for modem name

BufferLen

[in/out] Pointer to the buffer length; after the call it contains the real string length with the modem name. Attention: BufferLen must be selected large enough for the expected modem name before the function is called.

Return Values

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**).

Call Example

None

Requirements

	V5.6 ts_get_modem_name	
Windows:	95, ME, NT4, 2000, XP	
Header:	W95_S7.H, W95_S7M.H	
Library:	W95_S7.DLL, W95_S7M.DLL	

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\..
- ..\SIEMENS\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 realized 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 adapter cable) of the PG/PC.
- Configure the used PG/PC interface using the STEP 7 tool (S7EPATSX.EXE). The access point of the application **"S7ONLINE"** must be linked to the used parameter assignment of the module.
- In the event of a data link to S7-200 it is required to dial the relevant parameter assignment of the module 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.

A Appendix

A.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 MPI **
00CA	:	no resources available
00CE	:	module not found
00CF	:	driver not loaded
00E1	:	too many channels open
00E9	:	sin_serv.exe not started
00F1	:	no global dos storage available
0101	:	connection not established / configured
010A	:	negative acknowledgement received / timeout error
010C	:	data does not exist or disabled
0201	:	incorrect interface specified
0202	:	maximum amount of interfaces exceeded
0203	:	Toolbox already installed
0204	:	Toolbox already installed with other connections
0205	:	Toolbox not installed
0206	:	handle cannot be set
0207	:	data segment cannot be disabled
0209	:	incorrect data field
0302	:	block too small, DW does not exist
0303	:	block limit exceeded, correct amount
0310	:	module not found
0311	:	hardware error
0312	:	incorrect configuration parameters
0313	:	incorrect baud rate / interrupt vector
0314	:	incorrect HSA
0315	:	MPI address error
0316	:	HW device already allocated
0317	:	interrupt not available
0318	:	interrupt occupied
031A	:	connection error
0320	:	hardware error
0330	:	version conflict

0331	:	COM error
0332	:	no response
0333	:	COM error
0334	:	COM error
0336	:	no modem connection established
0337	:	job rejected
0381	:	device does not exist
0382	:	no driver or device found
0384	:	no driver or device found
03FF	:	system fault
4001	:	connection not known
4002	:	connection not established
4003	:	connection is being established
4004	:	connection broken down
800	:	Toolbox occupied
8001	:	not allowed in this operating status
8101	:	hardware error
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 capacity 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
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	:	adapter 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	:	adapter 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

A.2 Used Abbreviations

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