

# **PROFIBUS Application Program Interface**

## **User Manual**

Version 5.2  
Rev. 06

Date: 24-February-2003

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---

## 1 ABOUT THIS MANUAL

This user manual describes the PROFIBUS Application Program Interface and contains the following parts:

- Part 1: Software Installation
- Part 2: Driver Configuration
- Part 3: Application Program Interface
- Part 4: Basic Management Services
- Part 5: FMS Services
- Part 6: FM7 Services
- Part 7: DP Master Services
- Part 8: DP/V1-Master Services
- Part 9: FDL Services
- Part 10: DP-Slave Services
- Part 11: PROFIBUS Application Tools Library

## 2 RELATED PUBLICATIONS

This manual is written for the experienced C program developer and assumes the developer has a working knowledge of the PROFIBUS EN 50170/2 standard.

- PROFIBUS Data Link Layer
- PROFIBUS Application Layer (FMS, FM7 and LLI)
- PROFIBUS DP
- PROFIBUS DP/V1 (PROFIBUS Draft Specification V1.13 March 7,1997).
- PROFIBUS Implementation Guides Version 2.2

### **3 RELEASE NOTES**

- Version 5 contains the protocol components FAL (FMS, FM7), DP and FDL.
- From version 5.1 simultaneous operation of FAL, DP and FDL services is supported.
- From version 5.2 DP/V1 MSAC\_C2 client services are supported.
- The software release, version 5.21, offers the following features:
  - PROF104, Softing's PC/104 compliant board is supported
  - PROFcard is running with Windows 95 PC Card Services (PCMCIA Card- and Socket services), supporting Windows 95 Plug and Play.
  - The Windows NT PROFIBUS driver is supporting PROFboard, PROFcard and PROF104. Mixed usage of up to 10 boards of all three types in one PC is possible.
- The software releases, version 5.22 and version 5.23, offer the following features:
  - The PROFIBUS Windows drivers (Windows 2000, Windows NT, Windows ME/9x) are supporting the following boards:
    - PROFboard-ISA and PROFboard-PCI
    - PROFcard
    - PROF104 and PROF104-S
    - PROFgate, Softing's Ethernet gateway to PROFIBUS (only Windows 2000 / NT)Mixed usage of up to 10 boards of all six types in one PC is possible.
  - The PROFIBUS Windows drivers are now supporting the DP-Slave services
- The software release, version 5.24, supports the new PROFcard 2.
- The software releases, version 5.25 and version 5.26, offers the following new features:
  - The PROFIBUS Windows drivers are now running under Windows XP.
  - FG-300 PROFIBUS, the new Ethernet gateway to PROFIBUS is supported.

---

## Remarks on DP/V1

Simultaneous operation of DP/V1 and FAL (FMS, FM7) services is not supported.

## Remarks using PROFicard / PROFicard 2

Since PROFicard / PROFicard 2 is using Windows ME / 9x 32-Bit PC Card Services in Windows ME / 9x environment, DOS applications for PROFicard / PROFicard 2 do not run in Windows ME / 9x environment. It is recommended to use Windows 16/32 bit applications instead.

PROFicard / PROFicard 2 does not support DP-Slave services.

## Remarks using PROFI104-S

PROFI104-S does not support FMS/DPV1 services

## Remarks using FG300 PROFIBUS and PROFigate

The usage of up to 255 connections to FG-300 PROFIBUS- or PROFigate devices from one PC is possible.



# **PROFIBUS Application Program Interface**

## **Software Installation**

Version 5.2  
Rev. 06

Date: 24-February-2003

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

**COPYRIGHT NOTICE**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar



## CONTENTS

1 SCOPE .....	1
2 DELIVERABLES.....	1
3 SOFTWARE INSTALLATION .....	3
3.1 REQUIREMENTS AND PREPARATION.....	3
3.2 WINDOWS XP, WINDOWS 2000 AND WINDOWS NT .....	3
3.2.1 Installation procedure .....	3
3.2.2 Uninstall support.....	4
3.2.3 Directory structure and installed files .....	4
3.3 WINDOWS ME AND WINDOWS 9X .....	9
3.3.1 Installation procedure .....	9
3.4.2 Uninstall support.....	9
3.4.3 Directory structure and installed files .....	10
4 FIRMWARE UPDATE .....	14
4.1 WINDOWS XP, WINDOWS 2000 AND WINDOWS NT .....	14
4.2 WINDOWS ME AND WINDOWS 9X .....	14



## 1 SCOPE

This manual describes the installation of the PROFIBUS Application Program Interface (PAPI) software for Microsoft operating systems on a PC/AT or compatible computer.

## 2 DELIVERABLES

Deliverables include the PROFIBUS Application Program Interface with the corresponding device drivers, libraries and header files, sample programs in source code and (if necessary) the PROFIBUS protocol firmware.

The PROFIBUS Application Layer Interface supports access of the following PROFIBUS functionalities:

- PROFIBUS DP and DP/V1
- PROFIBUS DP-Slave
- PROFIBUS Application Layer (FAL) with FMS and FM7 (FAL Management)
- PROFIBUS FDL (Direct access to Data Link Layer services)

Softing offers seven types of PROFIBUS interface boards for PC/AT or compatible computers:

- PROFIboard-ISA, a short-size ISA-Bus controller for a 16-bit-ISA slot
- PROFIboard-PCI, a PCI-Bus controller for a 32-bit-PCI slot
- PROFI104 and PROFI104-S, 16-bit PC/104-compliant boards
- PROFIcard / PROFIcard 2, Type II PC Cards
- FG-300 PROFIBUS and PROFIgate, Fieldbus Access via Ethernet

For the operating systems Windows XP, Windows 2000, Windows NT, Windows ME and Windows 9x, there are only one package supporting all types of boards:

- PROFIBUS Application Program Interface (order no. PB-DMK)

The software package on the CD contains a installation program. Software and User Manual are put in a bundle as "PROFIBUS Documentation and Media Kit".

### 3 SOFTWARE INSTALLATION

#### 3.1 REQUIREMENTS AND PREPARATION

Hardware installation should be done prior to software installation. Please refer to the corresponding hardware user manuals for PROFIBoard (ISA, PCI), PROFI104, PROFICard / PROFICard 2 or FG-300 PROFIBUS / PROFigate. The system requirements and the mechanical installation procedure are described in detail in these documents.

#### 3.2 WINDOWS XP, WINDOWS 2000 AND WINDOWS NT

##### 3.2.1 Installation procedure

Perform the following steps to install the PROFIBUS Application Program Interface on your hard disk:

1. Always log on with administrator rights
2. Please close all applications, especially the Control Panel, during installation. If the Control Panel window is not closed the PROFIBUS control panel applet cannot be installed.
3. Insert the "PROFIBUS Application Program Interface" CD into the CD-ROM drive.
4. Start the installation program SETUP.EXE from the CD using the Windows Explorer or File Manager by simply double clicking the file SETUP.EXE.
5. Follow the online instructions of the installation program.

The installation program offers functions to

- Install the PROFIBUS Software Development Kit with:
  - PAPI and PBTOOLS dynamic link libraries (.DLL)
  - PAPI and PBTOOLS import libraries (.LIB)
  - PAPI and PBTOOLS header files (.h)
  - PAPI and PBTOOLS source files (.c, .rc)
  - Samples source and header files
- Install the PROFIBUS Runtime System with:
  - Windows Kernel mode device drivers.
  - PROFIBUS protocol firmware.
  - PAPI and PBTOOLS dynamic link libraries (DLL).
  - Control Panel Applet to configure the device drivers

After the **first** software installation the system has to be restarted to take the PROFIBUS device drivers in effect. Remove the installation diskette from the diskette drive and simply restart the system according to the on-screen instructions. When the system restarts, Windows will load the PROFIBUS device drivers and then open the PROFIBUS Control Panel Applet to configure the PROFIBUS device drivers.

### 3.2.2 Uninstall support

- To uninstall the PROFIBUS software, open the Control Panel and double-click the “Add / Remove Programs” icon.
- Select “PROFIBUS Runtime System” to remove it from computer.
- Select “PROFIBUS Software Development Kit” to remove it from the computer.

### 3.2.3 Directory structure and installed files

This section describes the directory structure of the PROFIBUS Runtime System and the PROFIBUS Software Development Kit after installation on hard disk.

#### 3.2.3.1 PROFIBUS Runtime System

The PROFIBUS Runtime System is installed in the following directories:

**<system directory>\SYSTEM32\DRIVERS**

PROFIBRD.SYS  
PROFIPRT.SYS  
PROFIPNP.SYS

**PROFIBUS kernel mode drivers**

PROFIBUS hardware driver  
PROFIBUS protocol driver  
PROFIBUS PnP hardware driver (only Windows XP and Windows 2000)

**<system directory>\SYSTEM32**

PBCPL.CPL  
PAPI.DLL  
PAPITH32.DLL  
  
PBTOOLS.DLL

**PROFIBUS operating system components**

PROFIBUS control panel applet  
PROFIBUS API dynamic link library  
PROFIBUS API 16Bit->32Bit thunk dynamic link library  
PROFIBUS Tools dynamic link library

**<common files directory>**

**\SOFTING\PROFIBUS\RTS\FIRMWARE**

\*.SBN

**PROFIBUS firmware files**

PROFIBUS firmware files

### 3.2.3.2 PROFIBUS Software Development Kit

The main directory of the PROFIBUS Software Development Kit (SDK) is installed in the selected installation directory and has the following directory structure:

- <installation directory>\SDK\DDB                    PROFIBUS device data base
- <installation directory>\SDK\DOC                   PROFIBUS documentation
- <installation directory>\SDK\PAPI                   PROFIBUS Application Program Interface
- <installation directory>\SDK\SAMPLES            PROFIBUS sample programs

#### PROFIBUS Device Data Base (DDB)

<b><u>.\SDK\DDB</u></b>		<b>PROFIBUS device data base</b>
	SOFTB203.GSD	DP-Master device description for PROFiboard and PROFi104
	SOFTB204.GSD	DP-Master device description for PROFicard / PROFicard 2
	SOFTB205.GSD	DP-Slave device description for PROFiboard, PROFi104 and PROFi104-S

#### PROFIBUS Documentation

<b><u>.\SDK\DOC</u></b>		<b>PROFIBUS documentation</b>
	PBDMKMAN.PDF	PROFIBUS Application Program Interface user manual

#### PROFIBUS Application Program Interface (PAPI)

<b><u>.\SDK\PAPI</u></b>		<b>PROFIBUS Application Program Interface</b>
<b><u>.\SDK\PAPI\INC_GLB</u></b>		<b>Global header files</b>
	KEYWORDS.H	Source file keywords
	PB_IF.H	Global PROFIBUS API function prototypes, constants and data types
	PB_TYPE.H	Compiler-independent standard data types
	PB_CONF.H	Constants for implementation and configuration
	PB_FMB.H	FMB service-specific constants and data types
	PB_FMS.H	FMS service-specific constants and data types
	PB_FM7.H	FM7 service-specific constants and data types
	PB_DP.H	DP-Master service-specific constants and data types
	PB_DPS.H	DP-Slave service-specific constants and data types
	PB_FDL.H	FDL service-specific constants and data types
	PB_ERR.H	Error constants and data types
	PB_NTDRV.H	PROFIBUS WinNT/Win2K drivers constants and data types

**.\SDK\PAPI\WIN32**

PAPI.LIB  
PAPI.DLL  
PAPITH32.DLL  
PBTOOLS.LIB  
PBTOOLS.DLL

**Libraries**

PAPI 32 Bit import library  
PAPI 32 Bit dynamic link library  
PAPI 16Bit->32Bit thunk dynamic link library  
PROFIBUS Tools 32 Bit import library  
PROFIBUS Tools 32 Bit dynamic link library

**.\SDK\PAPI\WIN16**

PAPI\_L.LIB  
PAPI\_L.DLL  
PBT\_L.LIB  
PBT\_L.DLL

**Libraries**

PAPI 16 Bit import library  
PAPI 16 Bit dynamic link library  
PROFIBUS Tools 16 Bit import library  
PROFIBUS Tools 16 Bitdynamic link library

**.\SDK\PAPI\SRC**

PAPI.C  
PAPIAUX.C  
FMBGDL.C  
FMSGDL.C  
FM7GDL.C  
DPGDL.C  
DPSGDL.C  
FDLGDL.C  
PAPI.RC  
PBTOOLS.RC

**Source files**

Global PAPI functions  
Global help functions  
FMB specific help functions  
FMS specific help functions  
FM7 specific help functions  
DP-Master specific help functions  
DP-Slave specific help functions  
FDL specific help functions  
PAPI resource file  
PBTOOLS resource file

BUSPARAM.C

Help functions to calculate the default bus parameters

CCRL.C

Help functions to calculate CRL resources

**.\SDK\PAPI\INC**

BUILDNR.PAPI.H  
BUILDNR.PBTOOLS.H  
RESOURCE.H  
VERSION.H

**Local header files**

PAPI version build number  
PBTOOLS version build number  
Common resource file  
Version string

**.\SDK\PAPI\DEF**

PAPI.DEF  
PBTOOLS.DEF

**Definition files**

PAPI definition file  
PBTOOLS definition file

**.\SDK\PAPI\MAKE**

PAPI.MAK  
PBTOOLS.MAK

**Generation files**

Make file for generating the PAPI dynamic link library  
Make file for generating PBTOOLS dynamic link library



## PROFIBUS Sample programs

### .\\SDK\\SAMPLES\\DPDEMONT

#### DP master sample program

#### .\\SDK\\SAMPLES\\DPDEMONT\\SRC

DPDEMONT.C	Sample main program
DPLAYER.C	DP master specific services
FMBLAYER.C	FMB specific services

#### .\\SDK\\SAMPLES\\DPDEMONT\\INC

DPDEMONT.H	Configuration data for DP master and the DP slaves
------------	----------------------------------------------------

#### .\\SDK\\SAMPLES\\DPDEMONT\\MAKE

DPDEMONT.MAK	Make file for generating DPDEMONT sample program
--------------	--------------------------------------------------

### .\\SDK\\SAMPLES\\DPSDEMONT

#### DP slave sample program

#### .\\SDK\\SAMPLES\\DPSDEMONT\\SRC

DPSDEMONT.C	Sample main program
DPSLAYER.C	DP slave specific services

#### .\\SDK\\SAMPLES\\DPSDEMO\\INC

DPSDEMONT.H	Global function prototypes
-------------	----------------------------

#### .\\SDK\\SAMPLES\\DPSDEMO\\MAKE

DPSDEMONT.MAK	Make file for generating DPSDEMONT sample program
---------------	---------------------------------------------------

### .\\SDK\\SAMPLES\\DPDEMO

#### DP master sample program

#### .\\SDK\\SAMPLES\\DPDEMO\\SRC

DPDEMO.C	Sample main program
----------	---------------------

#### .\\SDK\\SAMPLES\\DPDEMO\\INC

DPDEMO.H	Configuration data for DP master and the DP slaves
----------	----------------------------------------------------

#### .\\SDK\\SAMPLES\\DPDEMO\\MAKE

DPDEMO.MAK	Make file for generating DPDEMO sample program
------------	------------------------------------------------

### .\\SDK\\SAMPLES\\DPSDEMO

#### DP slave sample program

#### .\\SDK\\SAMPLES\\DPSDEMO\\SRC

DPSFRAME.C	Sample main program
DPSPROFI.C	DP slave specific services
DPSSCHED.C	Schedule functions

#### .\\SDK\\SAMPLES\\DPSDEMO\\INC

DPSFRAME.H	Global function prototypes
DPSPROFI.H	Global function prototypes
DPSSCHED.H	Global function prototypes

#### .\\SDK\\SAMPLES\\DPSDEMO\\MAKE

DPSDEMO.MAK      Make file for generating DPSDEMO sample program

### .\SDK\SAMPLES\DPV1DEMO

#### **DPV1 master sample program**

#### .\SDK\SAMPLES\DPV1DEMO\SRC

DPV1DEMO.C      Sample main program

#### .\SDK\SAMPLES\DPV1DEMO\INC

DPV1DEMO.H      Configuration data for DP master and the DP slaves

#### .\SDK\SAMPLES\DPV1DEMO\MAKE

DPV1DEMO.MAK      Make file for generating DPV1DEMO sample program

### .\SDK\SAMPLES\FMSDEMO

#### **FMS sample program**

#### .\SDK\SAMPLES\FMSDEMO\SRC

FMSDEMO.C      Sample program

#### .\SDK\SAMPLES\FMSDEMO\MAKE

FMSDEMO.MAK      Make file for generating FMSDEMO sample program

### .\SDK\SAMPLES\FDLDEMO

#### **FDL sample program**

#### .\SDK\SAMPLES\FMSDEMO\SRC

FDLDEMO.C      Sample program

#### .\SDK\SAMPLES\FMSDEMO\MAKE

FDLDEMO.MAK      Make file for generating FDLDEMO sample program

### .\SDK\SAMPLES\WIN32

#### **Executable sample programs**

DPDEMONT.EXE      DP master sample program (Win32-Interface)

DPSDEMONT.EXE      DP slave sample program (Win32-Interface)

DPDEMO.EXE      DP master sample program (PAPI)

DPSDEMO.EXE      DP slave sample program (PAPI)

DPV1DEMO.EXE      DPV1 master sample program (PAPI)

FMSDEMO.EXE      FMSDEMO sample program (PAPI)

FDLDEMO.EXE      FDLDEMO sample program (PAPI)

README.TXT      Notes on how to start the sample programs

### 3.3 WINDOWS ME AND WINDOWS 9X

#### 3.3.1 Installation procedure

Perform the following steps to install the PROFIBUS Application Program Interface on your hard disk:

1. Please close all applications, especially the Control Panel, during installation. If the Control Panel window is not closed the PROFIBUS control panel applet cannot be installed.
2. Insert the "PROFIBUS Application Program Interface" CD into the CD-ROM drive.
3. Start the installation program SETUP.EXE from the CD using the Windows Explorer or File Manager by simply double clicking the file SETUP.EXE.
4. Follow the online instructions of the installation program.

The installation program offers functions to

- Install the PROFIBUS Software Development Kit with:
  - PAPI and PBTOOLS dynamic link libraries (.DLL)
  - PAPI and PBTOOLS import libraries (.LIB)
  - PAPI and PBTOOLS header files (.h)
  - Samples source and header files
- Install the PROFIBUS Runtime System with:
  - Windows 95 virtual device driver.
  - PROFIBUS protocol firmware.
  - PAPI and PBTOOLS dynamic link libraries (DLL).
  - Control Panel Applet to configure the device drivers

After the **first** software installation the system has to be restarted to take the PROFIBUS device driver in effect. Remove the installation diskette from the diskette drive and simply restart the system according to the on-screen instructions. When the system restarts, Windows ME or Windows 9x will load the PROFIBUS device driver and then open the PROFIBUS Control Panel Applet to configure the PROFIBUS device drivers.

#### 3.4.2 Uninstall support

To uninstall the PROFIBUS software, open the Control Panel and double-click the "Software" icon.

- Select "PROFIBUS Runtime System" to remove it from the computer.
- Select "PROFIBUS Software Development Kit" to remove it from the computer

## 3.4.3 Directory structure and installed files

This section describes the directory structure of the PROFIBUS Runtime System and the PROFIBUS Software Development Kit after installation on hard disk.

### 3.4.3.1 PROFIBUS Windows Runtime System

The PROFIBUS Runtime System is installed in the following directories:

<system directory>\SYSTEM	PROFIBRD.VXD	<b>PROFIBUS virtual device driver</b> PROFIBUS virtual device driver
<system directory>\SYSTEM	PBCPL.CPL PROFIBRD.DLL PAPI.DLL PAPITH32.DLL PBTOOLS.DLL	<b>PROFIBUS operating system components</b> PROFIBUS control panel applet PROFIBUS Driver dynamic link library PROFIBUS API dynamic link library PROFIBUS API THUNK dynamic link library PROFIBUS Tools dynamic link library
<common files directory> SOFTING\PROFIBUS\RTS\FIRMWARE	*.SBN	<b>PROFIBUS firmware files</b> PROFIBUS firmware files

### 3.4.3.2 PROFIBUS Software Development Kit

The main directory of the PROFIBUS Software Development Kit (SDK) is installed in the selected installation directory and has the following directory structure:

- <installation directory>\SDK\DDB                      PROFIBUS device data base
- <installation directory>\SDK\DOC                      PROFIBUS documentation
- <installation directory>\SDK\PAPI                      PROFIBUS Application Program Interface
- <installation directory>\SDK\SAMPLES                      PROFIBUS sample programs

#### PROFIBUS Device Data Base (DDB)

<u>.\SDK\DDB</u>		<b>PROFIBUS device data base</b>
	SOFTB203.GSD	DP-Master device description for PROFIboard and PROFI104
	SOFTB204.GSD	DP-Master device description for PROFIcard / PROFIcard 2
	SOFTB205.GSD	DP-Slave device description for PROFIboard, PROFI104 and PROFI104-S

#### PROFIBUS Documentation

<u>.\SDK\DOC</u>		<b>PROFIBUS documentation</b>
	PBDMKMAN.PDF	PROFIBUS Application Program Interface user manual

#### PROFIBUS Application Program Interface (PAPI)

<u>.\SDK\PAPI</u>		<b>PROFIBUS Application Program Interface</b>
<u>.\SDK\PAPI\INC_GLB</u>		<b>Global header files</b>
	KEYWORDS.H	Source file keywords
	PB_IF.H	Global PROFIBUS API function prototypes, constants and data types
	PB_TYPE.H	Compiler-independent standard data types
	PB_CONF.H	Constants for implementation and configuration
	PB_FMB.H	FMB service-specific constants and data types
	PB_FMS.H	FMS service-specific constants and data types
	PB_FM7.H	FM7 service-specific constants and data types
	PB_DP.H	DP-Master service-specific constants and data types
	PB_DPS.H	DP-Slave service-specific constants and data types
	PB_FDL.H	FDL service-specific constants and data types
	PB_ERR.H	Error constants and data types

## **.\SDK\PAPI\WIN16**

PAPI_L.LIB	<b>Libraries</b>
PAPI_L.DLL	PAPI import library
PBT_L.LIB	PAPI dynamic link library
PBT_L.DLL	PROFIBUS Tools import library
	PROFIBUS Tools dynamic link library

## **.\SDK\PAPI\WIN32**

PAPI.LIB	<b>Source files</b>
PAPI.DLL	PAPI import library
PAPITH32.DLL	PAPI dynamic link library
PBTOOLS.LIB	PAPI thunk dynamic link library
PBTOOLS.DLL	PROFIBUS Tools import library
	PROFIBUS Tools dynamic link library

## **PROFIBUS Sample programs**

### **.\SDK\SAMPLES\DPDEMO**

#### **DP master sample program**

#### **.\SDK\SAMPLES\DPDEMO\SRC**

DPDEMO.C Sample main program

#### **.\SDK\SAMPLES\DPDEMO\INC**

DPDEMO.H Configuration data for DP master and the DP slaves

#### **.\SDK\SAMPLES\DPDEMO\MAKE**

DPDEMO.MAK Make file for generating DPDEMO sample program

### **.\SDK\SAMPLES\DPV1DEMO**

#### **DPV1 master sample program**

#### **.\SDK\SAMPLES\DPV1DEMO\SRC**

DPV1DEMO.C Sample main program

#### **.\SDK\SAMPLES\DPV1DEMO\INC**

DPV1DEMO.H Configuration data for DP master and the DP slaves

#### **.\SDK\SAMPLES\DPV1DEMO\MAKE**

DPV1DEMO.MAK Make file for generating DPV1DEMO sample program

### **.\SDK\SAMPLES\FMSDEMO**

#### **FMS sample program**

#### **.\SDK\SAMPLES\FMSDEMO\SRC**

FMSDEMO.C Sample program

#### **.\SDK\SAMPLES\FMSDEMO\MAKE**

FMSDEMO.MAK Make file for generating FMSDEMO sample program

### .\SDK\SAMPLES\FDLDEMO

#### **FDL sample program**

#### .\SDK\SAMPLES\FMSDEMO\SRC

FDLDEMO.C

Sample program

#### .\SDK\SAMPLES\FMSDEMO\MAKE

FDLDEMO.MAK

Make file for generating FDLDEMO sample program

### .\SDK\SAMPLES\WIN32

#### **Executable sample programs**

DPDEMO.EXE

DP master sample program

DPSDEMO.EXE

DP slave sample program

DPV1DEMO.EXE

DPV1 master sample program

FMSDEMO.EXE

FMSDEMO sample program

FDLDEMO.EXE

FDLDEMO sample program

README.TXT

Notes on how to start the sample programs

## **4 FIRMWARE UPDATE**

### **4.1 WINDOWS XP, WINDOWS 2000 AND WINDOWS NT**

The firmware update will be done automatically by the PROFIBUS hardware driver, if the version of the installed firmware file and the firmware on the PROFIBUS interface differ.

### **4.2 WINDOWS ME AND WINDOWS 9X**

Use the PROFIBUS control panel applet, located in the 'Control Panel' directory, to update the firmware onto the PROFIBUS interface. Open the PROFIBUS control panel applet by simply double click the "PROFIBUS" icon and select the PROFIBUS interface(s) and click the Update Firmware... button to download the current PROFIBUS firmware onto the PROFIBUS interface(s).



# **PROFIBUS Application Program Interface**

## **Driver Configuration**

Version 5.2  
Rev. 01

Date: 24-February-2003

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 - 4 56 56-0  
Fax (++49) 89 - 4 56 56-399

© Copyright by Softing AG, 2000-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 2000-2003 by Softing AG, Haar

---

---

## CONTENTS

1. GENERAL .....	1
2. HARDWARE RESOURCES .....	1
2.1 PROFIBOARD-ISA, PROFI104 AND PROF104-S.....	1
2.2 PROFICARD .....	1
2.3 PROFIBOARD-PCI .....	2
3. PROFIBUS CONTROL PANEL APPLET.....	3
3.1 OVERVIEW.....	3
3.1.1 PROFIBUS tree.....	4
3.1.2 Information area .....	4
3.1.3 Status bar .....	4
3.1.4 Buttons .....	5
3.2 SCAN NODES .....	5
3.3 UPDATE FIRMWARE .....	5
3.4 ADD- AND EDIT A PROFIBUS INTERFACE .....	5
3.4.1 Select Node Name .....	6
3.4.2 Select Operating Mode .....	7
3.4.3 PROFIboard-ISA, PROFI104 and PROFI104-S parameters.....	8
3.4.4 PROFIcard / PROFIcard 2 parameters.....	9
3.4.4.1 PROFIcard / PROFIcard 2 parameters using Windows XP, Windows 2000 or Windows ME/9x.....	9
3.4.4.2 PROFIcard / PROFIcard 2 parameters using Windows NT .....	10
3.4.4.2.1 PROFIcard / PROFIcard 2 software interface .....	10
3.4.4.2.2 PROFIcard / PROFIcard 2 (NT standard) .....	11
3.4.4.2.3 PROFIcard / PROFIcard 2 (Cardware).....	12
3.4.5 PROFIboard-PCI parameters.....	13
3.4.6 PROFIgate / FG-300 parameters.....	14
3.4.6.1 PROFIgate / FG-300 address.....	14
3.4.6.2 Timeout parameters for PROFIgate / FG-300 .....	15
3.5 REMOVE A PROFIBUS INTERFACE .....	16



### 1. GENERAL

Using Windows 32-Bit operating systems (Windows XP, Windows 2000, Windows NT or Windows ME/9x) the PROFIBUS device drivers must be configured before starting any PROFIBUS application.

Use the PROFIBUS control panel applet, located in the 'Control Panel' directory, to configure the PROFIBUS device drivers. The configuration procedure is explained in detail in chapter 3.

### 2. HARDWARE RESOURCES

#### 2.1 PROFIBOARD-ISA, PROFI104 AND PROF104-S

##### I/O port address

Each board requires 4 bytes of I/O port address space. The I/O port address of each board has to be selected by means of switches on the board prior to installing the hardware. See the hardware installation manual for more information.

##### Dual-port memory (DP-RAM)

Data exchange between board and PC occurs by means of a dual ported RAM (DP-RAM), which is mapped onto the upper memory area of the PC at a selectable location between the addresses 0xC8000 and 0xFFFFF. Each board requires 16 KBytes. This area must not be used by any other device. The address of this area will be configured during PROFIBUS device driver configuration.

##### Interrupt request lines

Each board requires one of the following ISA bus interrupt request lines (IRQ): 5, 10, 11, 12 or 15 for exclusive use. Because of the ISA bus architecture, each IRQ can be used only by one hardware device. The IRQ line for each board will be configured during PROFIBUS hardware driver configuration.

#### 2.2 PROFICARD

##### PC Card attribute memory (Windows NT)

The PROFICard enabled with the standard Windows NT mechanism needs 4 KBytes of the upper memory area of the PC at a selectable location between the addresses 0xC8000 and 0xFFFFF. This memory area is used to access the attribute memory of PROFICard. The address of this area will be configured during the PROFIBUS device driver configuration.

##### Dual-port memory (DP-RAM)

Data exchange between PROFICard and PC occurs by means of a dual ported RAM (DP-RAM), which is mapped onto the upper memory area of the PC at a selectable location between the addresses 0xC8000 and 0xFFFFF. PROFICard requires 16 KBytes. This area must not be used by any other device. The address of this area will be configured during the PROFIBUS device driver configuration.

### **Interrupt request lines**

PROFICard requires one ISA bus interrupt request line (IRQ) for exclusive use. Because of the ISA bus architecture, each IRQ can be used only by one hardware device. The PROFICard can use any IRQ line. The IRQ line will be configured during the PROFIBUS device driver configuration.

## **2.3 PROFIBOARD-PCI**

### **Dual-port memory (DP-RAM)**

Data exchange between PROFIBOARD-PCI and PC occurs by means of a dual ported RAM (DP-RAM), which is mapped automatically to a memory location in the 32-bit address space of the PC. PROFIBOARD-PCI requires 16 respectively 32 KBytes. The address space will be configured automatically loading the PROFIBUS device driver.

### **Interrupt request lines**

PROFIBOARD-PCI requires a PCI bus interrupt request line (IRQ) for shareable use. The IRQ line will be configured automatically loading the PROFIBUS device driver.

#### **NOTE:**

Use the "Windows Diagnostics" program included in Windows to determine free I/O ports, memory and IRQs for each PROFIBUS hardware installed.

#### **CAUTION:**

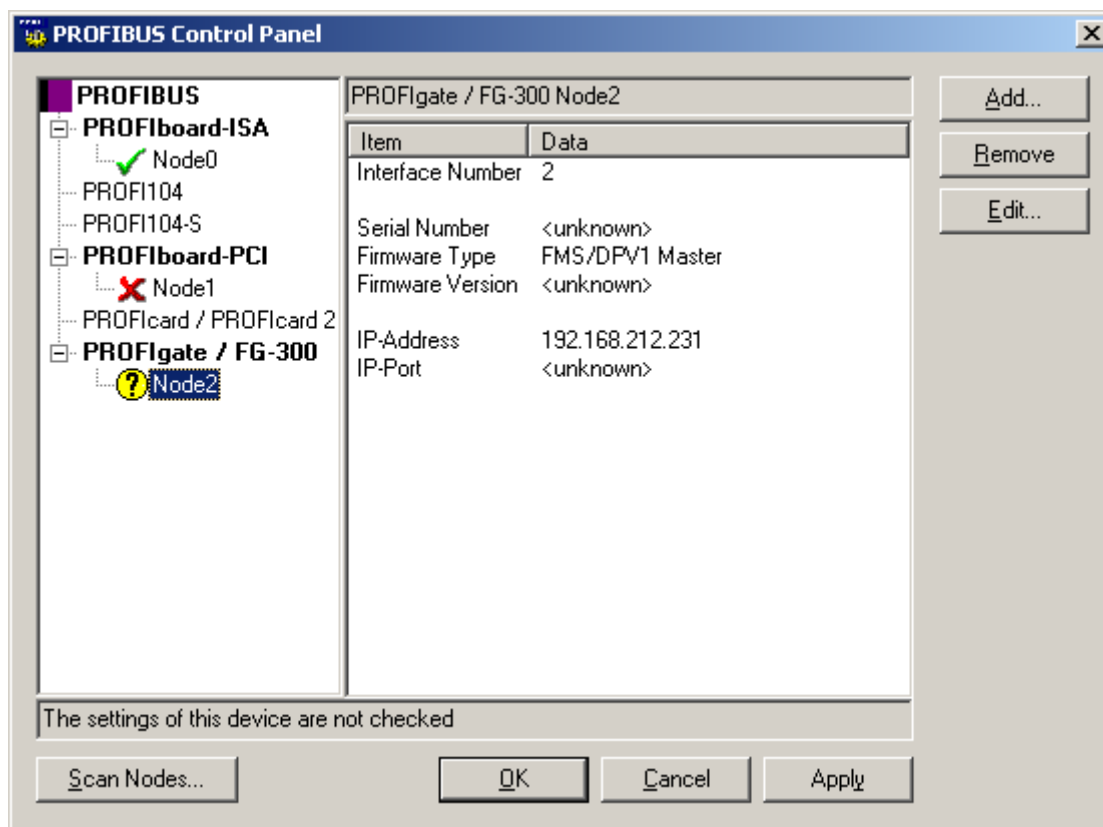
As with similar programs, the "Windows Diagnostics" can detect used IRQs and memory only if they are used in your current Windows installation by any kernel mode driver or by the kernel itself. If any hardware device is installed on your system (on the ISA bus or the motherboard) which uses a resource and this resource is not used by the system, it cannot be detected as "used IRQ" or "used memory" and, therefore, is displayed as "free". Use of such a resource for a PROFIBOARD may not function.

### 3. PROFIBUS CONTROL PANEL APPLET

Use the PROFIBUS control panel applet to configure the parameters of the PROFIBUS device driver. The PROFIBUS control panel applet is installed in the "Control Panel" during the installation procedure. To start the PROFIBUS control panel applet, simply double click the "PROFIBUS" icon.

#### 3.1 OVERVIEW

The user interface of the PROFIBUS control panel behaves like a standard Windows MDI (Multiple Document Interface) application. It consists of the PROFIBUS tree, push buttons and an information area where hardware specific configuration parameters will be displayed.



### 3.1.1 PROFIBUS tree




The PROFIBUS tree displays all configured PROFIBUS nodes with their current status subdivided into the following hardware (board) categories:

- PROFIboard-ISA (ISA version of PROFIboard with PROFIBUS FMS/DPV1 Master or DP Slave stack)
- PROFI104 (PC104 board with PROFIBUS FMS/DPV1 Master or DP Slave stack)
- PROFI104-S (PC104 board with PROFIBUS DP Slave stack)
- PROFIboard-PCI (PCI version of PROFIboard with PROFIBUS FMS/DPV1 Master or DP Slave stack)
- PROFIcard / PROFIcard 2 (PC card with PROFIBUS FMS/DPV1 Master stack)
- PROFIgate / FG-300 (PROFIBUS-Ethernet-Gateway with FMS/DPV1 Master or DP Slave stack)

These categories form the first level of the PROFIBUS tree and indicate the possible types of hardware interfaces. They appear at any time regardless whether such hardware interfaces are actually installed or not.

A PROFIBUS node is a PROFIBUS device acting as Master or Slave. PROFIboard-ISA, PROFIcard / PROFIcard 2, PROFI104 and PROFIgate / FG-300 can act as single node, PROFIboard-PCI has two channels and therefore can act as two nodes.

When a PROFIBUS node of a certain hardware category has been configured, the category will be displayed in bold letters and a node icon will appear below that category. A configured node is a node known to the PROFIBUS device driver. Depending on the state of the node, three different icons are used to display a node:

- The  icon indicates that the PROFIBUS node is working properly.
- The  icon indicates that the PROFIBUS node is not working properly.
- The  icon indicates that the PROFIBUS node is configured but not yet checked.

### 3.1.2 Information area

Main settings (firmware type, firmware version, resources, device names, ...) of the PROFIBUS node configuration are displayed in the information area.

### 3.1.3 Status bar

The current status of a PROFIBUS node is displayed in the status bar.



### 3.1.4 Buttons

The buttons in the PROFIBUS Control Panel have the following meaning:

<b>Add</b>	Click the <u>A</u> <i>dd</i> button to add a new PROFIBUS node to the PROFIBUS tree.
<b>Edit</b>	Click the <u>E</u> <i>dit</i> button to modify the settings of an existing PROFIBUS node in the PROFIBUS tree.
<b>Remove</b>	Click the <u>R</u> <i>emove</i> button to remove a PROFIBUS node from the PROFIBUS tree.
<b>Scan Nodes</b>	Click the <u>S</u> <i>can Nodes</i> button to search and configure automatically all plugged PROFIBUS interfaces (boards) in the PC (only Windows XP, Windows 2000 and Windows NT).
<b>Update Firmware</b>	Click the <i>Update Firmware</i> button to download the current PROFIBUS firmware onto the selected PROFIBUS interfaces (boards) in the PC (only Windows ME and Windows 9x).
<b>OK</b>	Click the <u>O</u> <i>K</i> button to save the current configuration settings and to load the device drivers
<b>Cancel</b>	Click the <i>Cancel</i> button to recover the last configuration settings and to load the device drivers
<b>Apply</b>	Click the <u>A</u> <i>pply</i> button to apply the configuration settings and load the device drivers

The procedure to add or edit a node or to scan automatically for nodes will be explained in detail in the following paragraphs.

### 3.2 SCAN NODES (ONLY WINDOWS XP, WINDOWS 2000 AND WINDOWS NT)

Clicking the S*can Nodes...* button the PROFIBUS hardware device driver searches for all PROFIBUS interfaces (boards) installed in your PC. All necessary resources like dual-ported-RAM, IO ports, interrupts, etc. will be determined automatically by the hardware device driver.

Depending on the hardware configuration, the scan process may take some time and it is indicated by a clock symbol.

### 3.3 UPDATE FIRMWARE (ONLY WINDOWS ME AND WINDOWS 9X)

Click the U*ppdate Firmware...* button to download the current PROFIBUS firmware onto the selected PROFIBUS interfaces (boards) in the PC.

### 3.4 ADD- AND EDIT A PROFIBUS INTERFACE

Click the A*dd* button to configure the hardware device driver for a new PROFIBUS interface.

At first, you will be asked to select the type of the interface board you want to configure.

Then, you may enter a symbolic name new node.

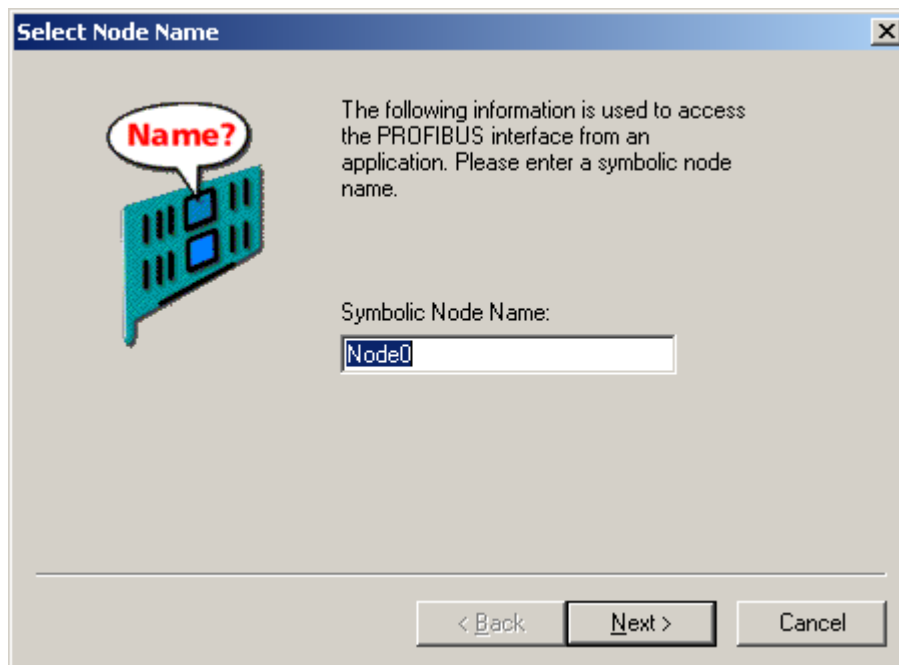
The remaining dialogs depend on the type of interface board. They are explained in detail in the sections below (3.4.1 to 3.4.6).

Using the *Edit* button an existing hardware device driver configuration of a PROFIBUS interface board can be modified. You may modify the node name and other parameter which depend on the type of the interface board.

The dialogs for modifying a node are the same as for adding a new node. They are explained in detail in the sections below (3.4.1 to 3.4.6).

### 3.4.1 Select Node Name (only Windows XP, Windows 2000 and Windows NT)

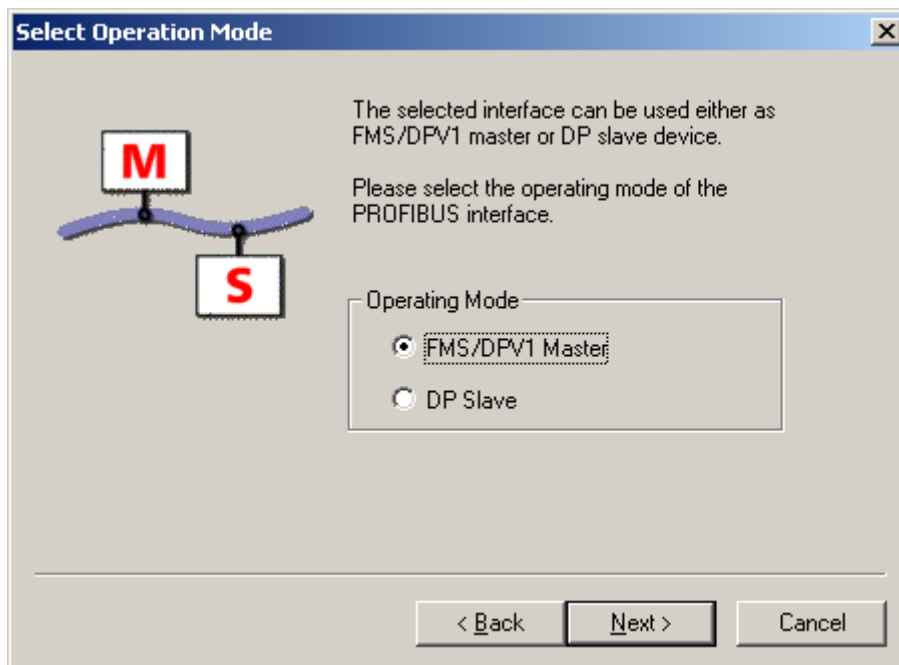
To access a logical PROFIBUS device from an application, a unique Win32 device name is required. In addition to the standard Win32 device name, assigned automatically by the PROFIBUS hardware device driver, a Win32 alias device name can be defined optionally.



Enter a symbolic node name in the edit field, click the *Next* button to continue the configuration. Click the *Cancel* button to abort the configuration and return to the main dialog.

### 3.4.2 Select Operating Mode

A PROFIBUS interface can operate either as FMS / DPV1 Master or as DP Slave device.



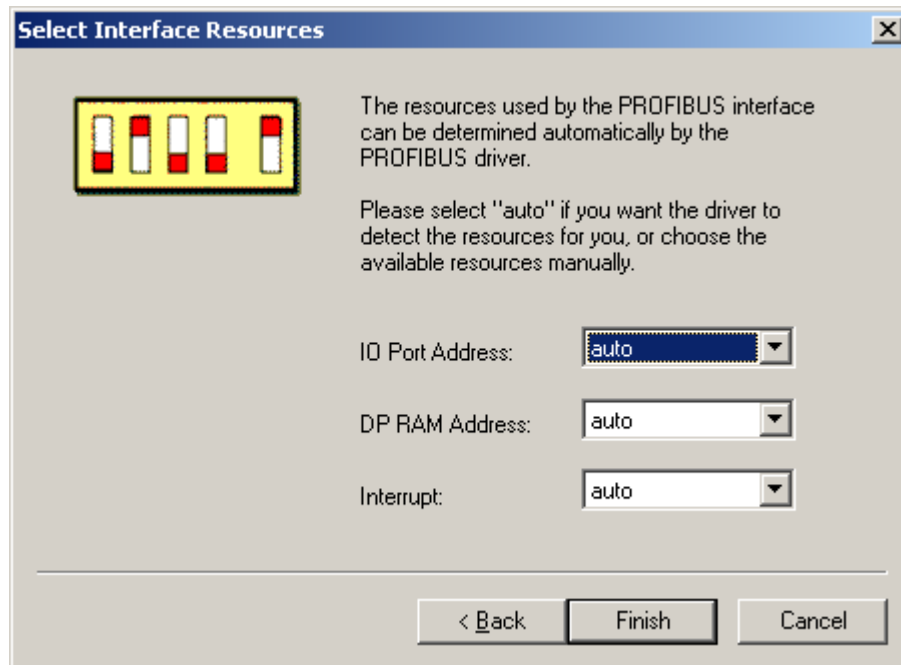
Select the desired operating mode, click the *Next* button to continue the configuration. Click the *Back* button to return to the previous dialog. Use the *Cancel* button to abort the configuration and to return to the main dialog.

**NOTE:**        **PROFicard / PROFicard 2** supports only FMS / DPV1 Master operating mode.  
                 **PROFI104-S** supports only DP Slave operating mode

**CAUTION:**    Changing the operating mode causes the hardware driver to download the selected firmware onto the PROFIBUS board. The existing firmware on the board will be overwritten!!!.

## 3.4.3 PROFIboard-ISA, PROFI104 and PROFI104-S parameters

Accessing PROFIboard-ISA, PROFI104 or PROFI104-S the PROFIBUS hardware device driver has to allocate resources like I/O ports, address space for the dual ported RAM and interrupt lines.



**IO-Port Address:** The I/O base address of the board. The auto-detection searches only the area between 0x200 and 0x3FC. To specify an address outside that range enter the base address in the edit box. Refer to the hardware installation manual for more information about valid I/O base addresses.

**DP RAM Address:** Address of the dual ported memory (see chapter 2.1 hardware resources).

**Interrupt:** IRQ line (see chapter 2.1 hardware resources).

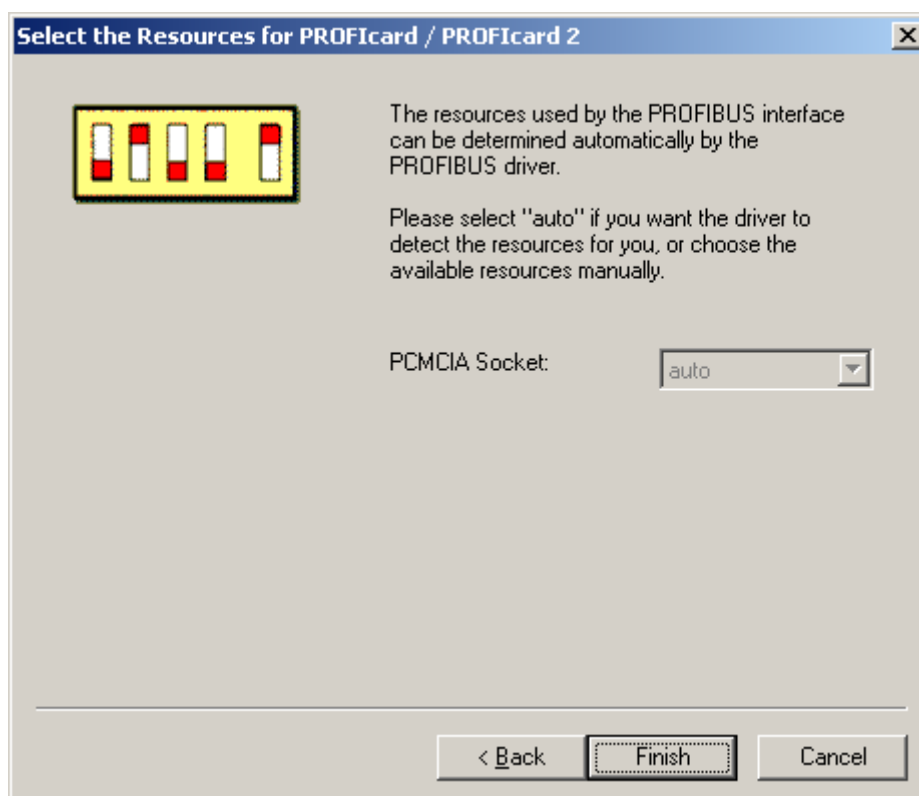
When selecting *auto* the PROFIBUS hardware driver determines the necessary resources of the PROFIBUS board.

Select the resources, click the *Finish* button to complete the configuration and return to the main dialog . Click the *Back* button to return to the previous dialog. Use the *Cancel* button to abort the configuration and to return to the main dialog.

### 3.4.4 PROFcard / PROFcard 2 parameters

#### 3.4.4.1 PROFcard / PROFcard 2 parameters using Windows XP, Windows 2000 or Windows ME/9x

Accessing PROFcard the PROFIBUS device driver has to allocate resources for the PC Card.



**PCMCIA Socket:** Socket number of PROFcard . This value is only needed to link an interface (board) number of the driver to the PC Card slot number where the PC Card is inserted. When a PROFcard is inserted into the PC Card slot, the driver searches the parameters for a PC Card configured for this socket.

Select the resources, click the Finish button to complete the configuration and to return to the main dialog. Click the Back button to return to the previous dialog. Use the Cancel button to abort the configuration and to return to the main dialog.

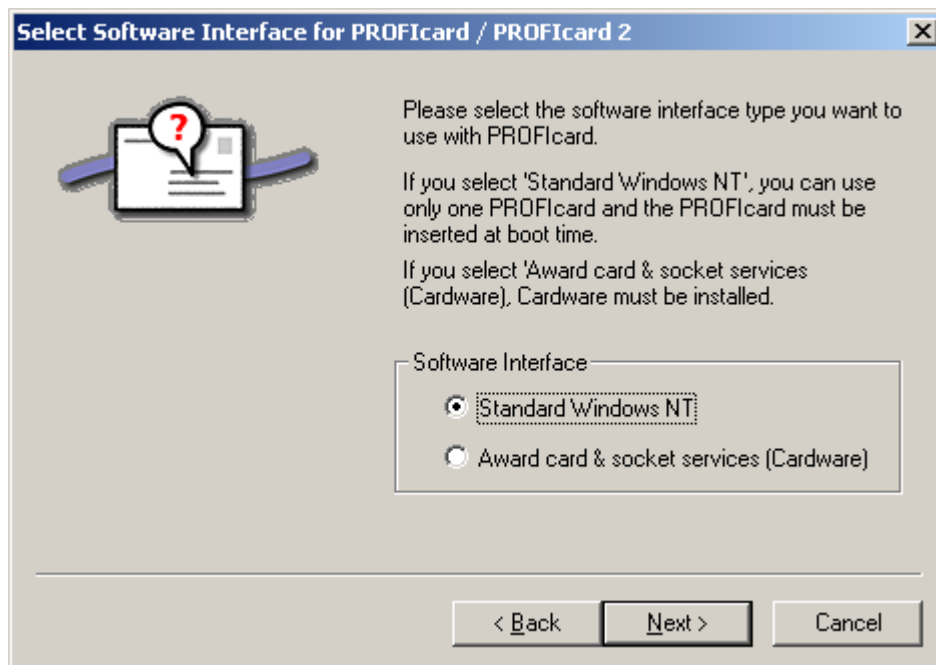
## 3.4.4.2 PROFcard / PROFcard 2 parameters using Windows NT

### 3.4.4.2.1 PROFcard / PROFcard 2 software interface

The PROFIBUS hardware device driver provides two mechanism for accessing PROFcard :

- The standard Windows NT enabling mechanism for PC Cards
- PC Card software "CardWare for Windows NT" by Award Software International Inc.

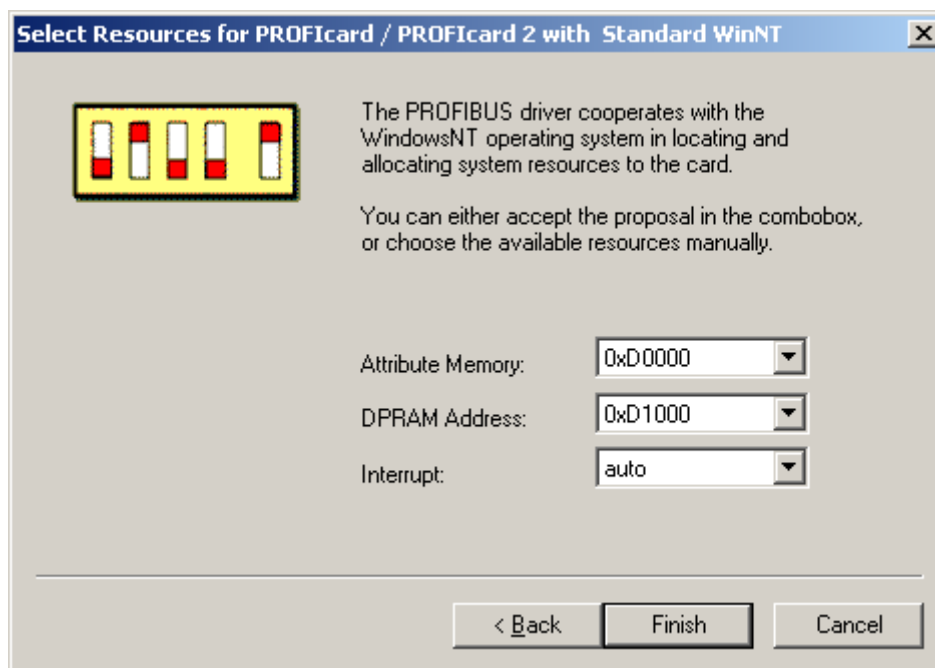
Using the standard Windows NT enabling mechanism for PC Cards only one PROFcard can be used. The PROFcard must be inserted at boot time.



Select the desired PC Card software interface, click the *Next* button to continue the configuration. Click the *Back* button to return to the previous dialog. Use the *Cancel* button to abort the configuration and to return to the main dialog.

### 3.4.4.2.2 PROFicard / PROFicard 2 (NT standard)

Accessing PROFicard using the Windows NT PC Card enabler the PROFIBUS hardware device driver has to allocate resources for PROFicard.



**Attribute Memory**      PC Card attribute memory (see chapter 2.2 hardware resources).

**DP RAM Address**      Address of the dual ported memory (see chapter 2.2 hardware resources).

**Interrupt**              IRQ line (see chapter 2.2 hardware resources).

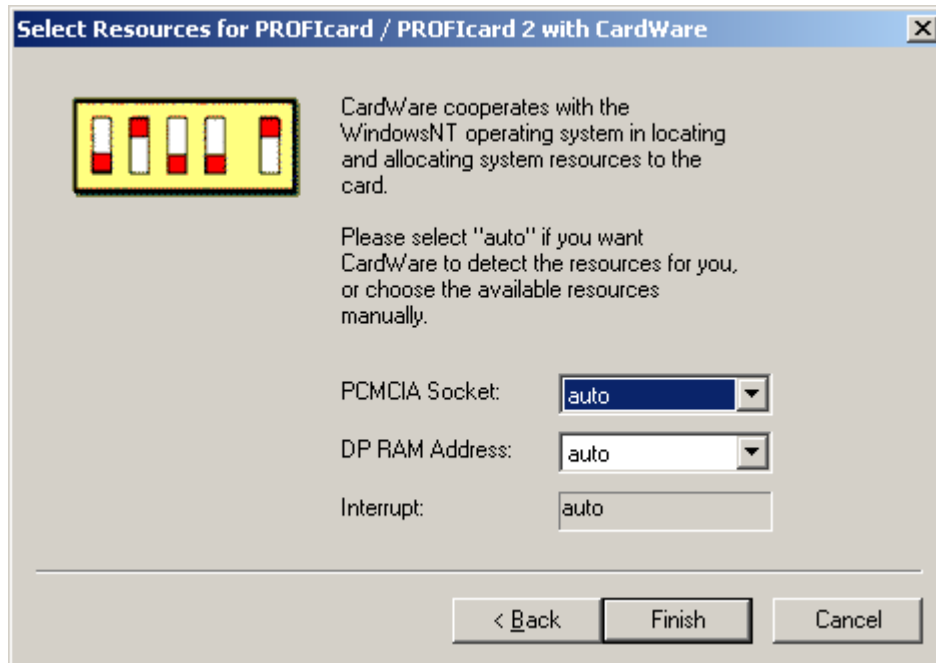
The PROFIBUS hardware device driver cannot provide auto detection of the resources, except the interrupt resource. You have to specify the physical values. You can configure only one "PROFicard (NT standard)", because the standard NT PC Card enabling mechanism supports only one PROFicard. You can't configure a mixture of both PROFicard solutions (PROFicard (NT standard) and PROFicard (Cardware)) with one installation.

Select the resources, click the *Finish* button to complete the configuration and return to the main dialog . Click the *Back* button to return to the previous dialog. Use the *Cancel* button to abort the configuration and to return to the main dialog.

**CAUTION:**      The Windows NT PC Card device driver starts during the boot process. Make sure to configure only free resources, because your system might not boot with an inserted card (e.g. if you have configured the PROFicard (NT standard) using the interrupt of the SCSI controller the system will hang-up during start-up).

## 3.4.4.2.3 PROFcard / PROFcard 2 (Cardware)

Accessing PROFcard using the "Award Cardware for Windows NT" the PROFIBUS hardware device driver has to allocate resources for PROFcard.



**PCMCIA Socket:** Socket number of PROFcard. This value is only needed to link an interface (board) number of the driver to the PC Card slot number where PROFcard is inserted. When a PROFcard is inserted into the PC Card slot, the driver searches the parameters for a PROFcard configured for this socket.

**DP RAM Address:** Address of the dual ported memory (see chapter 2.2 hardware resources).

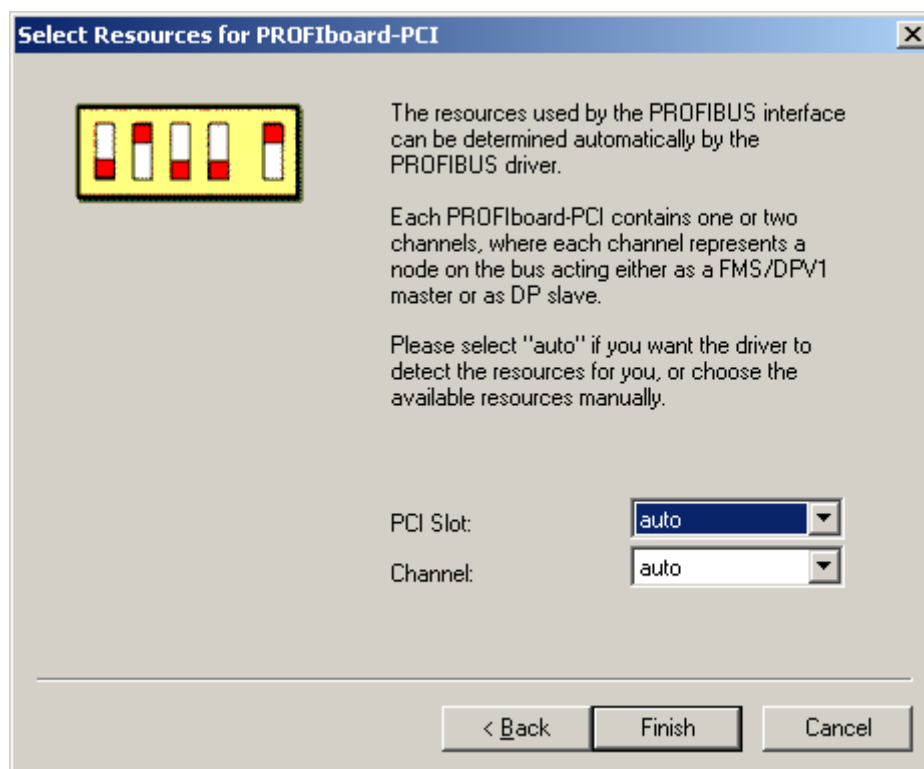
**Interrupt:** The PROFIBUS hardware driver determines automatically the IRQ line.

Select the resources, click the *Finish* button to complete the configuration and to return to the main dialog. Click the *Back* button to return to the previous dialog. Use the *Cancel* button to abort the configuration and to return to the main dialog.



### 3.4.5 PROFIboard-PCI parameters

A PROFIboard-PCI interface contains one or two channels, where each channel represents a node in the PROFIBUS network.



**PCI Slot**                      PCI slot number where the interface is plugged in.

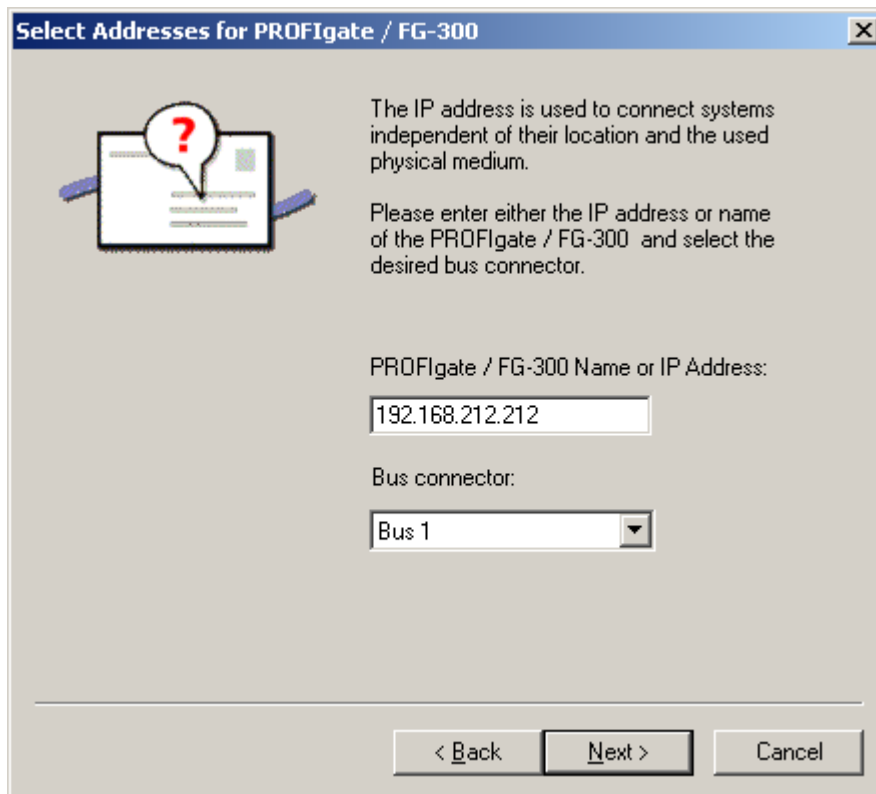
**Channel**                      Channel number (only dual channel PROFIboard-PCI board).

Click the *Finish* button to complete the configuration and return to the main dialog . Click the *Back* button to return to the previous dialog. Use the *Cancel* button to abort the configuration and to return to the main dialog.

## 3.4.6 PROFigate / FG-300 parameters

### 3.4.6.1 PROFigate / FG-300 address

The PROFIBUS Driver for Windows XP, Windows 2000 and Windows NT is capable of accessing a remote *PROFigate / FG-300* PROFIBUS-Ethernet-Gateway over a TCP/IP network and to treat it like a local PROFIBUS interface. This requires the IP address or the IP host name of PROFigate / FG-300 to be known to the driver.



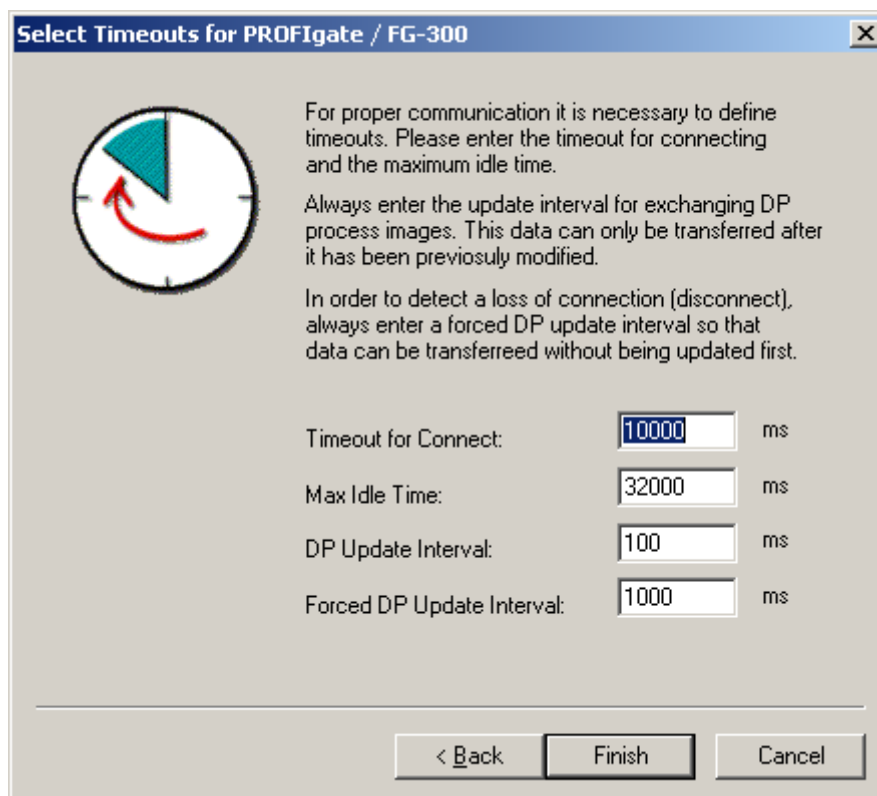
Enter the host name (if the IP network has a name server) or the IP address assigned to PROFigate / FG-300 in the edit field,

The FG-3000 PROFIBUS-Ethernet-Gateway contains up to three PROFIBUS interfaces, where each interface represents a node on the bus acting either as FMS/DPV1 master or as DP Slave. Select the desired PROFIBUS interface.

Click the *Next* button to continue the configuration. Click the *Cancel* button to abort the configuration and to return to the main dialog.

### 3.4.6.2 Timeout parameters for PROFIgate / FG-300

For proper communication with PROFIgate / FG-300 it is necessary to define some timeout values.



**Select Timeouts for PROFIgate / FG-300**

For proper communication it is necessary to define timeouts. Please enter the timeout for connecting and the maximum idle time.

Always enter the update interval for exchanging DP process images. This data can only be transferred after it has been previously modified.

In order to detect a loss of connection (disconnect), always enter a forced DP update interval so that data can be transferred without being updated first.

Timeout for Connect:  ms

Max Idle Time:  ms

DP Update Interval:  ms

Forced DP Update Interval:  ms

< Back Finish Cancel

<b>Timeout for Connect</b>	Time control interval for monitoring of connection establishment to PROFIgate / FG-300.
<b>Max Idle Time</b>	Time control interval to monitor the connection.
<b>DP Update Interval</b>	Update interval time for exchanging the DP process image. The process image will only be transferred if data have changed.
<b>Forced DP Update Interval</b>	Forced update interval time for exchanging the DP process image (regardless whether data have changed or not).

Enter the desired values, click the *Finish* button to complete the configuration and return to the main dialog . Click the *Back* button to return to the previous dialog. Using the *Cancel* button you can abort the configuration and return to the main dialog.

### **3.5 REMOVE A PROFIBUS INTERFACE**

Click the Remove button to remove a PROFIBUS interface from the PROFIBUS tree. After removing an interface click the Apply button to apply the new configuration and to reload the device drivers.

# **PROFIBUS Application Program Interface**

## **User Interface**

Version 5.2  
Rev. 05

Date: 24-February-2003

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---

## CONTENTS

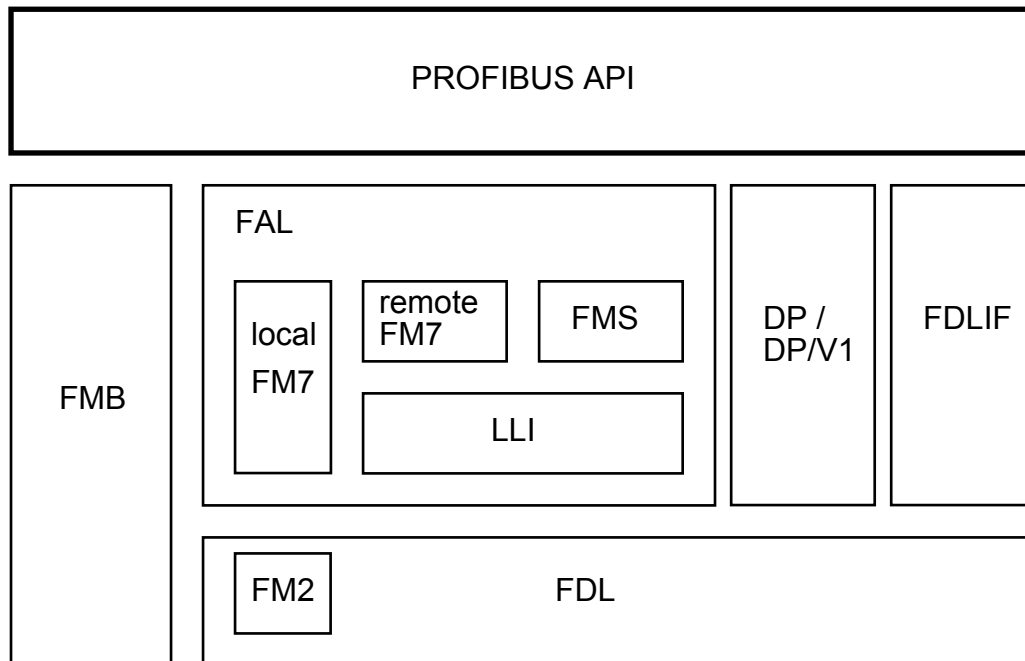
1 SCOPE .....	1
2 OVERVIEW .....	2
3 USER INTERFACE USING WINDOWS ME / 9X .....	3
3.1 INITIALIZATION AND SHUT DOWN .....	3
3.1.1 Profi-Init and Profi-Set-Default .....	3
3.1.2 Profi-End .....	4
3.2 SEND / RECEIVE INTERFACE .....	5
3.2.1 PROFIBUS Service Description Block .....	6
3.2.2 Profi-Snd-Req-Res .....	7
3.2.3 Profi-Rcv-Con-Ind .....	8
3.3 DATA INTERFACE .....	9
3.3.1 Profi-Set-Data .....	10
3.3.2 Profi-Get-Data .....	11
3.3.3 Profi-Set-Dps-Input-Data .....	12
3.3.4 Profi-Get-Dps-Input-Data .....	13
3.3.5 Profi-Get-Dps-Output-Data .....	14
3.4 ADDITIONAL INTERFACE FUNCTIONS .....	15
3.4.1 Profi-Ack-Irq .....	15
3.4.2 Profi-Get-Versions .....	16
3.4.3 Profi-Get-Serial-Device-Number .....	17
3.4.4 Profi-GetLast-Error .....	18
3.5 INTERFACE RETURN VALUES .....	19
4 USER INTERFACE USING WINDOWS XP, WINDOWS 2000 OR WINDOWS NT .....	21
4.1 LOGICAL DEVICES .....	22
4.1.1 Directory structure of logical devices .....	22
4.1.2 Low-level devices .....	23
4.1.3 Management devices .....	24
4.1.4 Data-oriented high-level devices .....	28
4.1.5 Service-oriented high-level devices .....	29
4.1.6 Access rights .....	32

4.2	PROGRAM INTERFACES .....	33
4.2.1	Data structures .....	34
4.3.1	CreateFile .....	36
4.3.2	CloseHandle .....	39
4.3.3	GetLastError .....	41
4.3.4	DeviceIoControl .....	43
4.3.5	ReadFile .....	50
4.3.6	ReadFileEx .....	53
4.3.7	WriteFile .....	55
4.3.8	WriteFileEx .....	58
4.3.9	GetOverlappedResult .....	60
4.3.10	SetFilePointer .....	62
4.3.11	FileIOCompletionRoutine .....	64
4.4	PROFIBUS APPLICATION PROGRAM INTERFACE .....	66
4.4.1	Initialization and Shut down .....	67
4.4.2	Send / Receive Interface .....	71
4.4.3	Data Interface .....	75
4.4.4	Additional Interface Functions .....	85
4.5	ENHANCED PROFIBUS APPLICATION PROGRAM INTERFACE .....	87
4.5.1	Profi-Open-Basic-Management .....	88
4.5.2	Profi-Open .....	90
4.5.3	Profi-Close .....	92
4.5.4	Profi-Write-Service .....	93
4.5.5	Profi-Read-Service .....	94
4.5.6	Profi-Read-Multi .....	97
4.5.7	Profi-Write-Data .....	100
4.5.8	Profi-Read-Data .....	102
4.5.9	Profi-Get-Cntrl-Info .....	104
4.5.10	Profi-Set-Timeout .....	105
4.5.11	Profi-Get-Timeout .....	107
4.5.12	Profi-Set-Queue-Size .....	108
4.5.13	Profi-Get-Queue-Size .....	109
4.5.14	Profi-Get-Overrun-Count .....	110
4.6	INTERFACE RETURN VALUES .....	111



## 1 SCOPE

This manual describes the common access functions to all components of Softing's PROFIBUS protocol software .



Depending on the component of the PROFIBUS protocol to be used, this document should be read in conjunction with one or more of the following parts of the PROFIBUS User Manual:

- "Basic Management"
- "FMS Services"
- "FM7 Services"
- "DP Services"
- "DP/V1 Services"
- "DP-Slave Services"
- "FDL Services"

## **2 OVERVIEW**

The services the PROFIBUS protocol (FMS, FM7 and DP / DP/V1) offers to the user are described in detail in EN 50170/2.

Those references describe the functions and parameters of the management and communication services, but do not provide instructions or structures for the programmer on the nuts and bolts of how to write an interface. This leaves room for implementations which can be adapted to optimally fit their respective system environment on the positive side, but which can also reduce the "openness" of the application layer interfaces.

Softing's PROFIBUS Application Program Interface (PAPI) provides two mechanisms for data exchange between application and protocol software and host and controller:

- a send/receive interface using request blocks for service-oriented data exchange and
- a data interface, which is used for fast cycle data exchange.

### 3 USER INTERFACE USING WINDOWS ME / 9X

#### 3.1 INITIALIZATION AND SHUT DOWN

The initialization functions **profi\_init** and **profi\_set\_default** are used to initialize the PROFIBUS Application Program Interface and to open the host driver.

##### 3.1.1 Profi-Init and Profi-Set-Default

The **profi\_init** or **profi\_set\_default** function is used to initialize the WIN32 PROFIBUS-API. The function has to be called before any other function of PROFIBUS-API is called.

INT16 profi\_init

```
(
    IN USIGN8          Board,
    IN UNSIGN32        ReadTimeout,
    IN UNSIGN32        WriteTimeout
);
```

INT16 profi\_set\_default

```
(
    IN USIGN8          Board,
    IN USIGN8          Channel,
    IN UNSIGN32        ReadTimeout,
    IN UNSIGN32        WriteTimeout
);
```

##### Function parameter description:

Board:                Number between 0..9 of the board to work on  
Channel:              PROFIBUS channel number (not supported).  
ReadTimeout:        ReceiveTimeout in msec (not supported).  
WriteTimeout:        Send Timeout in msec (not supported).

##### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(0)	Interface is initialized
- E_IF_INIT_INVALID_PARAMETER	(8)	Invalid initialize parameter
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond
- E_IF_INVALID_CNTRL_TYPE_VERSION	(11)	Invalid controller type or software version
- E_IF_NO_CNTRL_PRESENT	(28)	No controller present
- E_IF_INIT_FAILED	(31)	Initialization failed

Get detail information with **profi\_get\_last\_error** function

### 3.1.2 Profi-End

The **profi\_end** function is used to shut down the PROFIBUS API and the corresponding PROFIBUS interface.

```
INT16 profi_end  
(  
    VOID  
);
```

**Possible function return values:**

- E_OK	(0)	Shutdown executed successfully
- E_IF_EXIT_FAILED	(32)	shutdown executed not successfull, Get detail information with <b>profi_get_last_error</b> function

### 3.2 SEND / RECEIVE INTERFACE

The send/receive interface provides by means for both control flow and data flow between host and controller.

**Data flow** between the application and the communication is described by a service invariant and a large number of service specific data structures.

**Control flow** is directed by means of two functions, which control the data flow in both directions.

As in all communication protocols based on the OSI model, PROFIBUS distinguishes four application/communication interactions. These four basic interaction types are also called service primitives:

**Request:** PROFIBUS station A's application initiates a PROFIBUS service and passes the relevant parameters and data to the communication software.

**Indication:** The data passed by the request over the network have arrived at PROFIBUS station B (station A's communication partner for this service) and are passed to station B's application.

**Response:** For confirmed PROFIBUS services, station B's application must respond to the indication. Some services. eg. FMS-READ and GET-OD,. need additional response data.

**Confirmation:** For confirmed PROFIBUS services, station B's response is shared with station A's application by means of the "confirmation" service primitive.

There are then two service primitives, then namely request and response, by which an active (i.e. currently running) application starts up or responds to PROFIBUS services. The direction of control flow and data flow is from the application to the communication program.

In the case of the two other service primitives, indication and confirmation, data flow goes from communication to application. Control flow in this case depends on the communication/application synchronization mechanism.

The two cases described above are covered by two interface functions in the Softing PROFIBUS implementations.

The **profi\_snd\_req\_res** function is used for sending requests and responses. The **profi\_rcv\_con\_ind** function is used to poll for confirmations and indications.

### 3.2.1 PROFIBUS Service Description Block

For passing service data through the send/receive interface a service-independent SERVICE-DESCRIPTION-BLOCK and a service-specific DATA BLOCK.

The data structure T\_PROFI\_SERVICE\_DESCR describes the service to be performed by the protocol software.

Description of the service description block:

```
typedef struct T_PROFI_SERVICE_DESCR
{
    USIGN16      comm_ref;
    USIGN8       layer;
    USIGN8       service;
    USIGN8       primitive;
    INT8         invoke_id;
    INT16        result;
} T_PROFI_SERVICE_DESCR;
```

The service description block's elements are as follows:

- comm\_ref : Communication reference ("logical channel")
- layer: Layer instance the service invocation is directed to (FMS, FMB, FM7, DP, DPS, FDLIF, User)
- service\_id : Service to be performed in the instance specified in the layer.
- primitive : Service primitives (request, indication, response, confirmation)
- invoke\_id : ID to distinguish parallel service invocations
- result : Positive or negative result

The data block contains the service-specific data. Typically, for communication services these are data as described in the PROFIBUS EN 50170/2

Construction of the service-specific data blocks is described in the manuals FMS, FMB, FM7, FDLIF, DP, DPS and DP/V1.

### 3.2.2 Profi-Snd-Req-Res

The **profi\_snd\_req\_res** function is used to send a Service-Request or a Service-Response to the PROFIBUS interface.

INT16 profi\_snd\_req\_res

```
(
    IN T_PROFI_SERVICE_DESCR    FAR* pSdb,
    IN VOID                     FAR* pData,
    IN PB_BOOL                   Dummy
);
```

#### Function parameter description:

pSdb: Pointer to the data structure of type T\_PROFI\_SERVICE\_DESCR  
 pData: Pointer to service specific parameters and data  
 Dummy: Dummy parameter

#### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(0)	Function executed correctly
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_INVALID_LAYER	(12)	Invalid layer
- E_IF_INVALID_SERVICE	(13)	Invalid service identifier
- E_IF_INVALID_PRIMITIVE	(14)	Invalid service primitive
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_RESOURCE_UNAVAILABLE	(21)	No resource available
- E_IF_NO_PARALLEL_SERVICES	(22)	No parallel services allowed
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service temporarily not executable
- E_IF_SERVICE_NOT_SUPPORTED	(24)	Service not supported in subset
- E_IF_SERVICE_NOT_EXECUTABLE	(25)	Service not executable
- E_IF_INVALID_PARAMETER	(30)	Invalid parameter in REQ or RES
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized

### 3.2.3 Profi-Rcv-Con-Ind

The **profi\_rcv\_con\_ind** function is used to receive a Service-Indication or a Service-Confirmation from the PROFIBUS interface

```
INT16 profi_rcv_con_ind
(
    IN      T_PROFI_SERVICE_DESCR FAR* pSdb,
    IN      VOID                      FAR* pData,
    INOUT   USIGN16                  FAR* pDataLength
);
```

#### Function parameter description:

pSdb:	Buffer for service description block
pData:	Buffer for service specific data block
pDataLength:	On function invocation: maximal size of data block On function return: actual size of service specific data block

The function returns **CON\_IND\_RECEIVED** to signal that a confirmation or indication is available.

#### Possible function return values (defined in the header file PB\_ERR.H):

- NO_CON_IND_RECEIVED	(0)	There is no confirmation or indication
- CON_IND_RECEIVED	(1)	Confirmation or indication is available
- E_IF_FATAL_ERROR	(7)	Unrecoverable error on PROFIBUS controller, Error information stored in the data interface (see chapter 3.5)
- E_IF_INVALID_DATA_SIZE	(15)	Size of data block provided not sufficient
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized



### 3.3 DATA INTERFACE

In addition to the send / receive interface, the PROFIBUS Application Layer Interface offers a data interface which consists of data structures shared by host and controller. This data interface allows fast cyclic data transfer.

The data interface is performed by functions, which provide the data flow from and to the DPRAM area.

### 3.3.1 Profi-Set-Data

Using the **profi\_set\_data** function, shared data located in the DPRAM area can be written or modified.

```
INT16 profi_set_data
(
    IN USIGN8      DataId,
    IN USIGN16     Offset,
    IN USIGN16     DataSize,
    IN VOID        FAR* pData,
);
```

#### Function parameter description:

DataId: Identifier of the specified data structure in the Data Interface  
 Offset: Offset within the data structure  
 DataSize: Number of bytes to be written to the DPRAM  
 pData: Data block to be written

#### Possible values of data\_id (defined in the header file PB\_IF.H):

ID_DP_SLAVE_IO_IMAGE	0x80	Identifier of image for slave I/O data (DP)
ID_DP_STATUS_IMAGE	0x81	Identifier of image for status data (DP)

The structures of the data blocks are described in the service specific parts of the PROFIBUS User Manual.

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service currently not executable, access semaphore has been locked by the controller
- E_IF_SERVICE_NOT_SUPPORTED	(24)	Identifier is not supported
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized

### 3.3.2 Profi-Get-Data

The **profi\_get\_data** function is used to read shared data located in the DPRAM area.

```
INT16 profi_get_data
(
    IN    USIGN8      DataId,
    IN    USIGN16     Offset,
    INOUT USIGN16     FAR* pDataSize,
    OUT   VOID        FAR* pData,
);
```

#### Function parameter description:

DataId: Identifier of the specified data structure in the Data Interface  
 Offset: Offset within the data structure  
 pDataSize: On function invocation: maximal size of the data buffer (pData)  
 On function return: number of bytes actually read  
 pData: Pointer to data buffer

#### Possible values of data\_id (defined in the header file PB\_IF.H):

ID_DP_SLAVE_IO_IMAGE	0x80	Identifier of image for slave I/O data (DP)
ID_DP_STATUS_IMAGE	0x81	Identifier of image for status data (DP)
ID_EXCEPTION_IMAGE	0xF0	Identifier of image for exception data (IF)
ID_FW_VERS_IMAGE	0xF1	Identifier of image for firmware version (IF)
ID_SERIAL_DEVICE_NUMBER	0xF2	Identifier of image for serial device number (IF)

The structures of the data blocks are described in the service specific parts (IF, DP) of the manual.

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Size of data block provided not sufficient
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service currently not executable, access semaphore has been locked by the controller
- E_IF_SERVICE_NOT_SUPPORTED	(24)	Identifier is not supported
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized

### 3.3.3 Profi-Set-Dps-Input-Data

The **profi\_set\_dps\_input\_data** function writes the input data of the DP slave to the DP-Slave input data device. It always writes the full length of the data.

```
INT16 profi_set_dps_input_data
(
    IN    USIGN8*      pData,
    IN    USIGN8        DataLength,
    OUT   USIGN8*      pState
);
```

#### Function parameter description:

**pData:** Pointer to a USIGN8 variable containing the input data

**DataLength:** Number of input data to be written (in bytes). If the number does not correspond with the configured length of the input data, the error message 'E\_IF\_INVALID\_DATA\_SIZE' is returned.

**pState:** Pointer to the current input data status bit field with:

- DPS\_INPUT\_STATE\_FREEZE\_ENABLED  
The slave has enabled the function for freezing the inputs.
- DPS\_INPUT\_STATE\_FREEZE\_COMMAND  
A corresponding Global\_Control command was received. Since the last time the function **profi\_set\_dps\_input\_data** was called the input data have been taken over as the data to be transmitted from the slave to the master. A corresponding Global\_Control command for picking up the input data was received from the master. After the execution of this function the bit is reset automatically.

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Too much user data
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

### 3.3.4 Profi-Get-Dps-Input-Data

The **profi\_get\_dps\_input\_data** function reads the currently set inputs and the associated status of the DP slave from the DP-Slave input data device.

```
INT16 profi_get_dps_input_data
(
    OUT   USIGN8*    pData,
    INOUT USIGN8*    pDataLength,
    OUT   USIGN8*    pState
);
```

#### Function parameter description:

**pData:** Pointer to a USIGN8 variable array to read the inputs of the slave.

**pDataLength:** (IN) Pointer to a USIGN8 variable indicating the buffer size in bytes  
(OUT) Number of input data read

**pState:** Pointer to the current input data status bit field with:

- DPS\_INPUT\_STATE\_FREEZE\_ENABLED  
The slave has enabled the function for freezing the inputs.
- DPS\_INPUT\_STATE\_FREEZE\_COMMAND  
Since the last '**profi\_set\_dps\_input\_data**' a corresponding Global\_Control command has been received. The status is read-only. The bit will only be reset with the function **profi\_set\_dps\_input\_data**.

#### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	User buffer too small
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

### 3.3.5 Profi-Get-Dps-Output-Data

The **profi\_get\_dps\_output\_data** function reads the current outputs of the DP slave from the DP-Slave output data device.

```
INT16 profi_get_dps_output_data
(
    OUT  USIGN8*    pData,
    INOUT USIGN8*    pDataLength,
    OUT  USIGN8*    pState
);
```

#### Function parameter description:

pData:	Pointer to a USIGN8 variable array to read the outputs of the slave.
pDataLength:	(IN) Pointer to a USIGN8 variable indicating the buffer size in bytes (OUT) Number of output data read
pState:	Pointer to the current output data status bit field with:
-	<ul style="list-style-type: none"> <li>- DPS_OUTPUT_STATE_SYNC_ENABLED The function for freezing the outputs has been enabled.</li> <li>- DPS_OUTPUT_STATE_SYNC_COMMAND A corresponding Global_Control command was received. Since the last time the function <b>profi_get_dps_output_data</b> was called, a Sync command has been received upon which new output data have been made ready. The bit is cleared automatically after access.</li> <li>- DPS_OUTPUT_STATE_CLEAR_DATA The outputs are in failsafe state. A corresponding command was received from the master.</li> <li>- DPS_OUTPUT_STATE_VALID_DATA No transmission errors have occurred during data transmission from the master and user data are exchanged (no timeout or watchdog error).</li> <li>- DPS_OUTPUT_STATE_NEW_DATA New output data were received from the master. Since the last access via <b>profi_get_dps_output_data</b> function new data have been delivered (independent of the Sync command). With this bit you can prevent reusing old data. The bit is cleared after access.</li> <li>- DPS_OUTPUT_STATE_GLOBAL_CONTROL Since the last time the output data were read, a Global_Control command has been received. The bit is cleared as soon as the output data have been read.</li> </ul>

#### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	User buffer too small
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

## 3.4 ADDITIONAL INTERFACE FUNCTIONS

### 3.4.1 Profi-Ack-Irq

The **profi\_ack\_irq** function is used to acknowledge an interrupt received from the PROFIBUS interface and to return the type of interrupt (only 16-Bit PROFIBUS API).

```
USIGN8 profi_ack_irq  
(  
    VOID  
);
```

**Possible function return values** (defined in the header file PB\_IF.H):

- REQ_IRQ	(0xF0)	IND/CON interrupt is received
- ACK_IRQ	(0x00)	ACK interrupt of REQ/RES is received
- DP_SLAVE_IO_REQ_IRQ	(0xE0)	DP-SLAVE-IO IND/REQ IRQ is received
- DP_SLAVE_IO_ACK_IRQ	(0x0E)	DP-SLAVE-IO acknowledge IND/REQ IRQ is received
- DP_DATA_STOP_REQ_IRQ	(0xD0)	DP data transfer stop REQ IRQ is received
- DP_DATA_STOP_ACK_IRQ	(0x0D)	Data transfer stop Acknowledge IRQ is received
- EXCEPTION_REQ_IRQ	(0xB0)	Exception REQ IRQ is received

### 3.4.2 Profi-Get-Versions

The **profi\_get\_versions** function is used to read the version strings of the PROFIBUS API and PROFIBUS firmware.

```
extern INT16 profi_get_versions
(
    OUT    CSTRING FAR*    pPapiVersion,
    OUT    CSTRING FAR*    pFirmwareVersion
)
```

#### Function parameter description:

pPapiVersion:            Buffer to receive the PAPI version string  
pFirmwareVersion:       Buffer to receive the firmware version string

**NOTE:** The buffers have to be provided with a buffer size of VERSION\_STRING\_LENGTH (defined in the header file PB\_CONF.H) by the user.

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)



### 3.4.3 Profi-Get-Serial-Device-Number

The **profi\_get\_serial\_device\_number** function is used to read the serial device number of the PROFIBUS controller.

```
INT16 profi_get_serial_device_number  
(  
    OUT USIGN32 FAR* pSerialDeviceNumber  
);
```

#### Function parameter description:

pSerialDeviceNumber:            Serial device number

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized

### 3.4.4 Profi-GetLast-Error

The **profi\_get\_last\_error** function is used to return an additional error code for the interface errors E\_IF\_NO\_CNTRL\_PRESENT, E\_IF\_INIT\_FAILED and E\_IF\_EXIT\_FAILED.

```
INT16 profi_get_last_error  
(  
    VOID  
);
```

**Possible function return values** (defined in the header file PB\_ERR.H):

- E_NO_ADD_DETAIL	(0x00)	No additional error code
- E_PBDRV_XXX	(> 0x00)	Additional error code

### 3.5 INTERFACE RETURN VALUES

This chapter gives an overview of the user interface return values. All possible return values are described in the header files PB\_IF.H and PB\_ERR.H.

#### Overview of User Interface error codes and return values

Identifier	Value	Description
- E_OK	0	No error occurred
- NO_CON_IND_RECEIVED	0	No confirmation or indication available
- CON_IND_RECEIVED	1	Confirmation or indication was received
- E_IF_FATAL_ERROR	7	Unrecoverable error on board <sup>1)</sup>
- E_IF_INIT_INVALID_PARAMETER	8	Invalid initialization parameter
- E_IF_NO_CNTRL_RES	10	Controller does not respond
- E_IF_INVALID_CNTRL_TYPE_VERSION	11	Invalid controller type or invalid firmware version
- E_IF_INVALID_LAYER	12	Invalid layer
- E_IF_INVALID_SERVICE	13	Invalid service identifier
- E_IF_INVALID_PRIMITIVE	14	Invalid service primitive
- E_IF_INVALID_DATA_SIZE	15	Not enough CMI data block memory
- E_IF_INVALID_CMI_CALL	19	Invalid CMI call
- E_IF_CMI_ERROR	20	Error occurred in CMI
- E_IF_RESOURCE_UNAVAILABLE	21	No resource available
- E_IF_NO_PARALLEL_SERVICES	22	No parallel services allowed
- E_IF_SERVICE_CONSTR_CONFLICT	23	Service temporarily not executable
- E_IF_SERVICE_NOT_SUPPORTED	24	Service not supported
- E_IF_SERVICE_NOT_EXECUTABLE	25	Service not executable
- E_IF_INVALID_ACCESS	26	Invalid access to protocol software
- E_IF_NO_CNTRL_PRESENT	28	No controller present
- E_IF_INVALID_PARAMETER	30	Invalid parameter in REQ or RES
- E_IF_INIT_FAILED	31	Init. API or Controller failed
- E_IF_EXIT_FAILED	32	Exit API or Controller failed
- E_IF_PAPI_NOT_INITIALIZED	33	API not initialized

- 1) **NOTE:** If the interface error **E\_IF\_FATAL\_ERROR** is indicated, the User can read additional information about this error via the service interface function **profi\_rcv\_con\_ind** or data interface function **profi\_get\_data**:

## Read additional error information via **profi\_rcv\_con\_ind**:

### *Service-Description-Block for Indication:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_EXCEPTION
USIGN8	primitive	IND
INT8	invoke_id	0
INT16	result	POS

### *Data block for Indication:*

Data structure      T\_EXCEPTION

USIGN8	task_id	Task in wich the fatal system error is occurred
USIGN8	par1	Exception parameter 1
USIGN16	par2	Exception parameter 2
USIGN16	par3	Exception parameter 3

## Read additional error information via **profi\_get\_data**:

```
profi_get_data (ID_EXCEPTION_IMAGE,      /* Identifier of the exception description */
               0,                        /* Offset in the exception description */
               (USIGN16 FAR*) &data_len, /* Size of the exception description */
               (T_EXCEPTION FAR*) &exception /* Pointer to the exception description */
               );
```

```
T_EXCEPTION      exception;               /* Defined in the header file PB_ERR.H */
USIGN16          data_descr_len = sizeof(T_EXCEPTION);
```

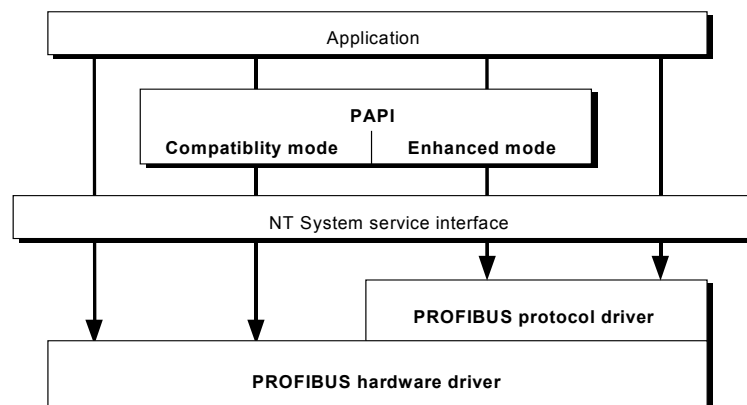
#### 4 USER INTERFACE USING WINDOWS XP, WINDOWS 2000 OR WINDOWS NT

The PROFIBUS driver for Windows XP, Windows 2000 and Windows NT offers access to the functionality of the PROFIBUS protocol stack which runs on the PC boards PROFIboard (ISA, PCI), PROFIcard / PROFIcard 2 and PROFI104 and on the PROFIBUS-Ethernet gateway PROFIgate / FG-300.

The software consists of the following parts:

- A low-level kernel device driver which provides hardware access to PROFIboard (ISA, PCI), PROFIcard and PROFI104 and remote access via Ethernet to PROFIgate / FG-300.
- An intermediate kernel device driver which supports access to PROFIBUS in a protocol-specific manner. The following devices exist: FDL SAP devices, DP Slave devices and FMS CR devices.
- An interface library (PAPI - PROFIBUS Application Program Interface) which provides access to the complete functionality of the drivers, and, in addition, offers a compatibility mode interface to simplify porting of existing PROFIBUS applications to Windows XP, Windows 2000 or Windows NT.

The software is designed to access up to ten PROFIBUS boards.



## 4.1 LOGICAL DEVICES

The kernel device drivers create basic logical devices during system startup and additional logical devices by special system calls. Low-level devices are provided to access the functionality of the low-level device driver, and high-level devices (management devices, FMS CR devices, FDL SAP devices, DP service devices and DP-Master data devices) are provided to access the functionality of the protocol device driver.

All Low-level devices and all management devices will be created when the device drivers are started. All the devices (e.g. FMS\CrZ, FDL\SapZ, DP\ServiceZ, DP\SlaveDataZ, DP\MSACZ where Z is an enumerated number) will be created by requests of the application program.

The logical devices are accessed by the I/O functions of Windows 2000 or Windows NT. Each logical device must be opened before it can be used. The devices are accessed by read, write, and control functions. After using a device, it should always be closed.

All read requests can be executed only if the device is open for read access. All write requests can be executed only if the device is open for write access. The access rights needed for control functions depend on the specific control function code.

### 4.1.1 Directory structure of logical devices

To access a logical device, a unique Win32 device name is required. Optionally a Win32 alias device name can be defined via the PROFIBUS control panel. The device names are created by the kernel device drivers.

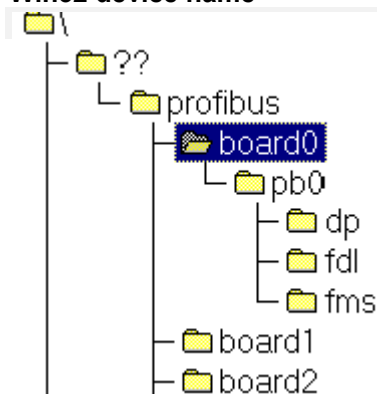
**Note:** The Win32 subsystem of Windows XP, Windows 2000 or Windows NT requires that all device names begin with the characters "\\.\".

The logical devices are structured hierarchically similar to a directory structure:

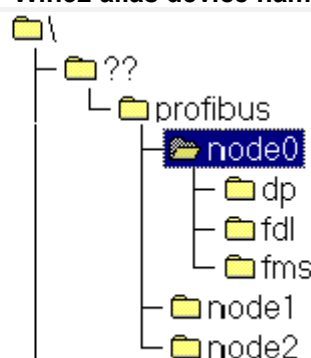
Example to access the **general service device** of board 0:

- Access via Win32 device name:           \\.\PROFIBUS\Board0\Pb0\Service
- Access via Win32 alias device name:   \\.\PROFIBUS\Node0\Service

**Win32 device name**



**Win32 alias device name**



**NOTES:**

In C programs, each backslash must be typed as "\\". Consequently, this device name in a C program is "\\.\PROFIBUS\Board0\Pb0\Service " or "\\.\PROFIBUS\Node0\Service".

The device names of all devices described below use these variables:

Y is the board number, 0..9

Z is an index, defining the special communication reference, DP slave or service access point

**4.1.2 Low-level devices****4.1.2.1 Board device**

Win32 Name: \\.\PROFIBUS\BoardY\Board

Win32 Alias Name: \\.\PROFIBUS\<SymbolicNodeName>\Board

Function: Services concerning the complete board

Read: Read the version information of the PROFIBUS protocol firmware

Ioctl: IOCTL\_PROFI\_GET\_DATA\_IMAGE  
Get a data image from the board

**4.1.2.2 General service device**

Win32 Name: \\.\PROFIBUS\BoardY\Pb0\Service

Win32 Alias Name: \\.\PROFIBUS\<SymbolicNodeName>\Service

Function: Read and write any PROFIBUS frame

Read: Read any received frame

Write: Write any frame

Ioctl: IOCTL\_PROFI\_SET\_TIMEOUT

IOCTL\_PROFI\_GET\_TIMEOUT

Set or read the time-outs for read/write operations to/from this device.

'Set' requires read access to the device.

'Get' can be done with any access to the device

The default time-out values of the General Service Device are 0 ms for read and write.

If writing of a frame fails with one of these error codes (E\_IF\_NO\_PARALLEL\_SERVICES, E\_IF\_RESOURCE\_UNAVAILABLE, E\_IF\_SERVICE\_CONSTR\_CONFLICT), the hardware driver retries to write the frame until either the write succeeds or the write time-out elapsed.

**NOTE:**

The **FMB\_EXCEPTION** indication will be generated by the firmware if a fatal error has been detected. The PROFIBUS hardware will be reinitialized.

If a **FMB\_RESET** or **FMB\_EXIT** confirmation has been received, the PROFIBUS hardware will be reinitialized.

#### **4.1.2.3 General DP-Master data device**

Win32 Name:            \\.\PROFIBUS\Board\Y\Pb0\DpData  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\DpData

Function:             Read and write of any DP data

Read:                 Read any DP data, not restricted to a slave  
Write:                Write any DP data, not restricted to a slave  
Ioctl:                 Set / Clear DP data bits in I/O data image, not restricted to a slave

#### **4.1.2.4 General DP-Slave input data device**

Win32 Name:            \\.\PROFIBUS\Board\Y\Pb0\DpSlaveInputData  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\DpSlaveInputData

Function:             Read and write of any DP-Slave input data

Read:                 Read DP-Slave input data and current status  
Ioctl:                 Write DP-Slave input data, read current status

#### **4.1.2.5 General DP-Slave output data device**

Win32 Name:            \\.\PROFIBUS\Board\Y\Pb0\DpSlaveOutputData  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\DpSlaveOutputData

Function              Read DP-Slave output data

Read:                 Read DP-Slave output data and current status

### **4.1.3 Management devices**

All management devices support the IoControl IOCTL\_PROFI\_READ\_MULTI to check more than one device for received frames.



#### 4.1.3.1 Basic management device

Win32 Name:            \\.\PROFIBUS\Board\YPb0\Management

Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\Management

Function:             Basic Management functionality

Read:                 Read FMB confirmations and indications

Write:                Write FMB requests and responses

Ioctl:                IOCTL\_PROFI\_CREATE\_FMS\_MANAGEMENT  
                       IOCTL\_PROFI\_CREATE\_FMS\_CR  
                       IOCTL\_PROFI\_CREATE\_FDL\_MANAGEMENT  
                       IOCTL\_PROFI\_CREATE\_FDL\_SAP  
                       IOCTL\_PROFI\_CREATE\_DP\_MANAGEMENT  
                       IOCTL\_PROFI\_CREATE\_DP\_MSAC  
                       IOCTL\_PROFI\_CREATE\_DP\_SLAVE\_DATA  
                       IOCTL\_PROFI\_CREATE\_DP\_SERVICE  
                       Create logical devices for selected SAPs, CRs, DP-Slaves  
                       Possible with any access to the device

Ioctl:                IOCTL\_PROFI\_SET\_QUEUE\_SIZE  
                       IOCTL\_PROFI\_GET\_QUEUE\_SIZE  
                       Set the maximum number of frames to buffer in the device driver  
                       Possible with any access to the device

Ioctl:                IOCTL\_PROFI\_GET\_OVERRUN\_COUNT  
                       Read and reset the overrun count  
                       Possible with any access to the device

Ioctl:                IOCTL\_PROFI\_SET\_TIMEOUT  
                       IOCTL\_PROFI\_GET\_TIMEOUT  
                       Set and read the time-outs for read from and write to all service-oriented, high-level  
                       devices of the selected PROFIBUS channel.  
                       'Set' requires read access to the device  
                       'Get' can be done with any access to the device  
                       The default time-out values of the high-level service-oriented devices are 30 ms for  
                       write and 7 seconds for read.

PROFIBUS services supported on this device:

FMB_SET_CONFIGURATION	FMB_LSAP_STATUS
FMB_SET_BUSPARAMETER	FMB_GET_LIVE_LIST
FMB_READ_BUSPARAMETER	FMB_FM2_EVENT
FMB_SET_VALUE	FMB_EXIT
FMB_READ_VALUE	FMB_EXCEPTION

#### NOTE:

**FMB\_SET\_CONFIGURATION must be the first service executed by the master application.**

**4.1.3.2 DP management device**

Win32 Name:            \\\\.\\PROFIBUS\\Board Y\\Pb0\\DP\\Management  
Win32 Alias Name:     \\\\.\\PROFIBUS\\<SymbolicNodeName>\\DP\\Management  
  
Function:             PROFIBUS DP management  
  
Read:                 Read DP management frames  
Write:                Write DP management frames  
Ioctl:                IOCTL\_PROFI\_CREATE\_DP\_MSAC  
                      IOCTL\_PROFI\_CREATE\_DP\_SLAVE\_DATA  
                      IOCTL\_PROFI\_CREATE\_DP\_SERVICE  
                      Create logical device for selected DP-Slave  
                      Possible with any access to the device

All DP Indications are received on this device.

PROFIBUS services supported on this device:

DP_DOWNLOAD_LOC	DP_GET_CFG
DP_UPLOAD_LOC	DP_SLAVE_DIAG
DP_START_SEQ_LOC	DP_RD_INP
DP_END_SEQ_LOC	DP_RD_OUTP
DP_GET_SLAVE_DIAG	DP_SET_SLAVE_ADD
DP_SET_PRM_LOC	DP_DOWNLOAD
DP_GET_MASTER_DIAG_LOC	DP_UPLOAD
DP_GET_SLAVE_PARAM	DP_START_SEQ
DP_INIT_MASTER	DP_END_SEQ
DP_ACT_PARAM_LOC	DP_ACT_PARAM
DP_DATA_TRANSFER	DP_GET_MASTER_DIAG
DP_EXIT_MASTER	DP_GLOBAL_CONTROL

Services not supported:

DP_SET_PRM	DP_DATA_EXCHANGE
DP_CHK_CF	

### 4.1.3.3 FMS management device

Win32 Name:            \\.\PROFIBUS\Board\YPb0\FMS\Management  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\FMS\Management

Function:             PROFIBUS FMS management

Read:                 Read FMS management frames  
Write:                Write FMS management frames  
Ioctl:                IOCTL\_PROFI\_CREATE\_FMS\_CR  
                      Create logical device for selected CR  
                      Possible with any access to the device

PROFIBUS services supported on this device:

FMS_OD_READ_LOC	FM7_LOAD_CRL_LOC
FMS_INIT_LOAD_OD_LOC	FM7_TERM_LOAD_CRL_LOC
FMS_LOAD_OD_LOC	FM7_SET_VALUE_LOC
FMS_TERM_LOAD_OD_LOC	FM7_READ_VALUE_LOC
FMS_CREATE_VFD_LOC	FM7_LSAP_STATUS_LOC
FMS_VFD_SET_PHYS_STATUS_LOC	FM7_IDENT_LOC
FMS_PI_SET_STATE_LOC	FM7_EVENT
FM7_READ_CRL_LOC	FM7_EXIT
FM7_INIT_LOAD_CRL_LOC	

Services not supported:

FM7\_RESET  
FM7\_SET\_BUSPARAMETER  
FM7\_READ\_BUSPARAMETER  
FM7\_GET\_LIVE\_LIST

#### **4.1.3.4 FDL management device**

Win32 Name:            \\.\PROFIBUS\Board\YPb0\FDL\Management  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\FDL\Management

Function:             PROFIBUS FDL management:

Read:                 Read FDL management frames  
Write:                Write FDL management frames  
Ioctl:                IOCTL\_PROFI\_CREATE\_FDL\_SAP  
                      Create logical device for selected SAP  
                      Possible with any access to the device

PROFIBUS service supported on this device:

FDLIF\_EXIT

Services not supported:

FDLIF\_SET\_BUSPARAMETER  
FDLIF\_READ\_BUSPARAMETER  
FDLIF\_EVENT

#### **4.1.4 Data-oriented high-level devices**

##### **4.1.4.1 DP slave data device**

Win32 Name:            \\.\PROFIBUS\Board\YPb0\DP\SlaveDataZ  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\DP\SlaveDataZ

Function               Read and write PROFIBUS DP data

Open:                 The device can be opened only if the slave is available through the PROFIBUS network.

Read:                 Read the DP Data of slave Z  
                      If the DP protocol is in the state STOP or the state flag of the slave is not zero, the last valid data is read, and the error code (E\_IF\_INVALID\_DP\_STATE or E\_IF\_SLAVE\_ERROR) will be returned.

Write:                Write the DP data of slave Z  
                      If the DP protocol is in the state STOP or the state flag of the slave is not zero, the data is copied to a buffer, and the (E\_IF\_INVALID\_DP\_STATE or E\_IF\_SLAVE\_ERROR) error code will be returned.

### 4.1.5 Service-oriented high-level devices

All service-oriented high-level devices support the IoControl IOCTL\_PROFI\_READ\_MULTI to check more than one device if frames have been received.

#### 4.1.5.1 DP service device

Win32 Name:            \\.\PROFIBUS\Board\YPb0\DP\ServiceZ  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\DP\ServiceZ

Function               Slave specific DP services

Read:                  Read slave-specific DP frames  
Write:                 Write slave-specific DP frames

PROFIBUS services supported on this device:

DP\_GET\_CFG  
DP\_SLAVE\_DIAG  
DP\_RD\_INP  
DP\_RD\_OUTP  
DP\_GLOBAL\_CONTROL

Services not supported:

DP\_SET\_PRM  
DP\_CHK\_CFG  
DP\_DATA\_EXCHANGE

#### 4.1.5.2 DP master slave acyclic device (DP/V1 device)

Win32 Name:            \\.\PROFIBUS\Board\YPb0\DP\MSACZ  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\DP\MSACZ

Function:              PROFIBUS DP/V1 Services

Read:                  Read a frame from slave Z  
Write:                 Write a frame to slave Z  
Close:                 A DP\_ABORT request will be sent automatically to close the Communications Reference (CR). This is an unconfirmed service.

PROFIBUS services supported on this device:

DP\_INITIATE  
DP\_READ  
DP\_WRITE  
DP\_ABORT

The maximum amount of opened MSAC devices is currently limited to 30. A Communication Reference (CR) need not be specified in the service descriptor block. The protocol driver handles the CRs used for MSAC devices.

#### **4.1.5.3 FDL SAP device**

Win32 Name:            \\.\PROFIBUS\Board Y\Pb0\FDL\SapZ  
Win32 Alias Name:     \\.\PROFIBUS\<SymbolicNodeName>\FDL\SapZ  
  
Function:              PROFIBUS FDL services  
  
Read:                  Read a frame from SAP Z  
Write:                 Write a frame to SAP Z  
Close:                 A FDLIF\_SAP\_DEACTIVATE request will be sent by the driver automatically to deactivate the SAP. This is a confirmed service.

PROFIBUS services supported on this device:

FDLIF\_SDA  
FDLIF\_SDN  
FDLIF\_SRD  
FDLIF\_REPLY\_UPDATE  
FDLIF\_REPLY\_UPDATE\_MULTIPLE  
FDLIF\_SAP\_ACTIVATE  
FDLIF\_RSAP\_ACTIVATE  
FDLIF\_SAP\_CHANGE\_ACCESS  
FDLIF\_SAP\_DEACTIVATE

Any indication received at a SAP device which is not open for read access will be ignored.

#### 4.1.5.4 FMS Communication Reference (CR) device

Win32 Name:            \\.\PROFIBUS\Board\YPb0\FMS\CrZ  
Win32 Alias Name:       \\.\PROFIBUS\<SymbolicNodeName>\FMS\CrZ

Function:               PROFIBUS FMS services

Read:                  Read a frame from CR Z  
Write:                  Write a frame to CR Z  
Close:                  A FMS\_ABORT request with code USER\_DISCONNECT will be sent automatically to close the Communication Reference (CR). This is an unconfirmed service.

**NOTE:**

Remote management indications are received at CR1.

PROFIBUS services supported on this device:

FMS_INITIATE	FMS_INFO_RPT
FMS_STATUS	FMS_UNSOLICITEDSTATUS
FMS_IDENTIFY	FMS_EVN_NOTIFY
FMS_READ	FMS_INFO_RPT_WITH_TYPE
FMS_WRITE	FMS_EVN_NOTIFY_WITH_TYPE
FMS_GET_OD	FMS_ABORT
FMS_READ_WITH_TYPE	FMS_REJECT
FMS_WRITE_WITH_TYPE	
FMS_DEF_VAR_LIST	FMS_OD_READ_LOC
FMS_DEL_VAR_LIST	FMS_INIT_LOAD_OD_LOC
FMS_INIT_DOWNL_SEQ	FMS_LOAD_OD_LOC
FMS_DOWNL_SEQ	FMS_TERM_LOAD_OD_LOC
FMS_TERM_DOWNL_SEQ	FMS_CREATE_VFD_LOC
FMS_INIT_UPL_SEQ	FMS_VFD_SET_PHYS_STATUS_LOC
FMS_UPL_SEQ	FMS_PI_SET_STATE_LOC
FMS_TERM_UPL_SEQ	FMS_VAR_DATA_EVENT_LOC
FMS_REQ_DOM_DOWNL	
FMS_REQ_DOM_UPL	FMS_GEN_INIT_DOWNL_SEQ
FMS_PI_CREATE	FMS_GEN_DOWNL_SEQ
FMS_PI_DELETE	FMS_GEN_TERM_DOWNL_SEQ
FMS_PI_START	
FMS_PI_STOP	FM7_INITIATE
FMS_PI_RESUME	FM7_ABORT
FMS_PI_RESET	FM7_READ_CRL_REM
FMS_PI_KILL	FM7_INIT_LOAD_CRL_REM
FMS_ALT_EVN_CND_MNT	FM7_LOAD_CRL_REM
FMS_ACK_EVN_NOTIFY	FM7_TERM_LOAD_CRL_REM
FMS_PHYS_READ	FM7_SET_VALUE_REM
FMS_PHYS_WRITE	FM7_READ_VALUE_REM
FMS_INIT_PUT_OD	FM7_LSAP_STATUS_REM
FMS_PUT_OD	FM7_IDENT_REM
FMS_TERM_PUT_OD	

**NOTE:**

If an FMS\_INITIATE request is received at a CR which is not opened for read access, the driver will automatically send a response with code E\_FMS\_INIT\_USER\_DENIED.

All other indications at a CR which is not opened for read access will be ignored.

#### **4.1.6 Access rights**

All management devices can be opened only once for read access and only once for write access.

The DP master slave acyclic devices may be opened several times for one slave with any access.

All other service-oriented devices (General Service Device, FMS CR Devices, FDL SAP Devices, DP Service Devices) may be opened any times for write access, but only once for read access.

All data-oriented devices (General Data Device, DP Slave Data Devices) may be opened any times for read access, but only once for write access.

All management devices and all service-oriented high-level devices use the General Service Device to communicate with the board. The data-oriented high-level devices (DP Slave Data Devices) additionally use the General Data Device. Therefore, the low-level devices can be used only if no management device or high-level device is open.

The board device may be opened any times for read access (to read the firmware version information), but only once for write access. It may be opened for write access only if there is no other device open on this board.



### 4.2 PROGRAM INTERFACES

The PROFIBUS driver for Windows 2000 and Windows NT supports three different program interfaces. Each of these interfaces has its own advantages (+) and disadvantages (-). The following list should help you decide which interface to use.

- **WIN32 System Interface**

The Win32 system interface consists of the standard Win32 system calls for device handling. These functions can be used to access the devices provided by the low-level and the protocol kernel mode device drivers.

- + Provides the whole functionality of the device drivers
- + Asynchronous read or write calls possible
- Most difficult to program

- **PROFIBUS Application Program Interface**

The PROFIBUS API provides the same set of functions as the PROFIBUS API for Windows ME, Windows 9x, Windows 3.x or MS-DOS. This interface does not use the functionality provided by the protocol device driver. All PROFIBUS API calls are implemented based on the functionality provided by the low-level device driver.

- + Same functions as in older PROFIBUS software. Use this interface if you want to port an existing application.
- One process can handle only one board.
- No functionality of the protocol driver

- **PROFIBUS Enhanced Application Program Interface**

The enhanced PROFIBUS API contains new functions in the PAPI which use the functionality of the protocol device driver.

- + Encapsulates the whole functionality of the protocol device driver
- + Provides additional functionality (frame size calculation)

### 4.2.1 Data structures

All program interfaces provide access to the service-oriented devices. A PROFIBUS frame consists of a service-independent description and a service-specific service-specific data block with parameters and data.

The data structure T\_PROFI\_SERVICE\_DESCR describes the service to be performed by the protocol software.

Description of the service description block:

```
typedef struct T_PROFI_SERVICE_DESCR
{
    USIGN16      comm_ref;
    USIGN8       layer;
    USIGN8       service;
    USIGN8       primitive;
    INT8         invoke_id;
    INT16        result;
} T_PROFI_SERVICE_DESCR;
```

The service description block's elements are as follows:

- comm\_ref : Communication reference ("logical channel")
- layer: Layer instance the service invocation is directed to (FMS, FMB, FM7, DP, DPS, FDLIF, FMS\_USR, FMB\_USR, FM7\_USR, DP\_USR, DPS\_USR, FDLIF\_USR)
- service\_id : Service to be performed in the instance specified in the layer.
- primitive : Service primitives (request, indication, response, confirmation)
- invoke\_id : ID to distinguish parallel service invocations
- result : Positive or negative result

The data block contains the service-specific data. Typically, for communication services these are data as described in the PROFIBUS EN 50170/2

Construction of the service-specific data blocks is described in the manuals FMS, FMB, FM7, FDLIF, DP and DP/V1.

### 4.3 PROFIBUS WIN32 SYSTEM INTERFACE

The following Win32 system calls can be used to handle the PROFIBUS devices. These calls are subsequently described in detail:

<b>CreateFile</b>	Open a PROFIBUS device
<b>CloseHandle</b>	Close a PROFIBUS device
<b>GetLastError</b>	Get the error code of the last failed system call
<b>DeviceIOControl</b>	Send a control code to a PROFIBUS device
<b>ReadFile</b>	Read data from a PROFIBUS device
<b>ReadFileEx</b>	Read data asynchronously from a PROFIBUS device
<b>WriteFile</b>	Write data to a PROFIBUS device
<b>WriteFileEx</b>	Write data asynchronously to a PROFIBUS device
<b>SetFilePointer</b>	Set the file pointer of a general data device
<b>FileIOCompletionRoutine</b>	Callback routine for asynchronous data transfer

## 4.3.1 CreateFile

The **CreateFile** function opens a PROFIBUS device and returns a handle that can be used to access the object.

HANDLE CreateFile

```
(
    LPCTSTR          lpFileName,
    DWORD            dwDesiredAccess,
    DWORD            dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes
    DWORD            dwCreationDistribution,
    DWORD            dwFlagsAndAttributes,
    HANDLE           hTemplateFile
);
```

### Function parameter description:

**lpFileName:** Points to a null-terminated string that specifies the name of the PROFIBUS device to open.

**dwDesiredAccess:** Specifies the type of access to the device. An application can obtain read access, write access or read-write access. Use the following flag constants to build a value for this parameter. Both **GENERIC\_READ** and **GENERIC\_WRITE** must be set to obtain read/write access. If **dwDesiredAccess** is 0, neither read nor write access is allowed; only **IOControl** operations that do not need a specific access right can be performed on the device.

Value	Meaning
<b>GENERIC_READ</b>	Specifies read access to the device. Data can be read from the device, and the file pointer can be moved.
<b>GENERIC_WRITE</b>	Specifies write access to the device. Data can be written to the device, and the file pointer can be moved.

**dwShareMode:** Set of bit flags that specifies how the device can be shared. If **dwShareMode** is 0, the device cannot be shared. No other open operations can be performed on the device. This flag cannot extend the constraints described in chapter 4.1.6 (Access rights).To share the device, use a combination of one or more of the following values:

Value	Meaning
<b>FILE_SHARE_READ</b>	Other open operations can be performed on the device for read access.
<b>FILE_SHARE_WRITE</b>	Other open operations can be performed on the device for write access.

lpSecurityAttributes: Always should be NULL; security is not supported by the PROFIBUS device drivers.  
 dwCreationDistribution: Must be OPEN\_EXISTING.  
 dwFlagsAndAttributes: Specifies the file attributes and flags for the file.

Attribute	Meaning
FILE_ATTRIBUTE_NORMAL	The file has no other attributes set. This attribute is valid only if used alone.
FILE_FLAG_OVERLAPPED	<p>Instructs the operating system to initialize the device so <b>ReadFile</b> and <b>WriteFile</b> operations that take a significant amount of time to process return ERROR_IO_PENDING. When the operation is finished, an event is set to the signaled state.</p> <p>When you specify FILE_FLAG_OVERLAPPED, the <b>ReadFile</b> and <b>WriteFile</b> functions must specify an <b>OVERLAPPED</b> structure: i.e. when FILE_FLAG_OVERLAPPED is specified, an application must perform overlapped reading and writing.</p> <p>General Data Device: When FILE_FLAG_OVERLAPPED is specified, the file position must be passed as part of the lpOverlapped parameter (pointing to an <b>OVERLAPPED</b> structure) to the <b>ReadFile</b> and <b>WriteFile</b> functions.</p> <p>This flag also enables more than one operation to be performed simultaneously with the handle (a simultaneous read and write operation, for example).</p>

hTemplateFile: This value must be NULL.

### Possible function return values

- If the function succeeds, the return value is an open handle to the specified device.
- If the function fails, the return value is INVALID\_HANDLE\_VALUE. To obtain extended error information, call **GetLastError**.

### NOTES:

You can use the **CreateFile** function to open a logical PROFIBUS device. The function returns a handle to the device. This handle can be used with the **ReadFile**, **WriteFile**, and **DeviceIOControl** function.

The **CloseHandle** function is used to close a handle returned by **CreateFile**.

## Example

```
#include <windows.h>
...
{
    HANDLE hBoard;
    ULONG   ErrorCode;

    hBoard = CreateFile  ("\\\\.\\PROFIBUS\\Board0\\Board"           // Name of the device
                        GENERIC_READ,                             // Access mode
                        FILE_SHARE_READ                           // Share mode
                        NULL,                                       // Pointer to securitydescriptor
                        OPEN_EXISTING,                             // How to create
                        FILE_ATTRIBUTE_NORMAL,                     // File attribute
                        NULL,                                       // Handle to template file
                        );

    if (hBoard == INVALID_HANDLE_VALUE)
    {
        // do error handling
        ErrorCode = GetLastError();
        ...
    }
    ...
}
```

### 4.3.2 CloseHandle

The **CloseHandle** function closes an open handle.

```
BOOL CloseHandle  
(  
    HANDLE      hObject  
);
```

#### Function parameter description:

hObject:           Identifies an open device handle.

#### Possible function return values

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

#### NOTES:

**CloseHandle** invalidates the specified object.

Use **CloseHandle** to close handles returned by calls to the **CreateFile** function.

Closing an invalid handle raises an exception. This includes closing a handle twice and not checking the return value and closing an invalid handle.

## Example

```
{  
    HANDLE hDevice;  
  
    // Open device  
    ULONG  ErrorCode;  
  
    hDevice = CreateFile (....  
  
    // do input / output  
    ...  
  
    // close device  
    if (!CloseHandle (hDevice))  
    {  
        // do error handling  
        ErrorCode = GetLastError();  
        ...  
    }  
}
```



### 4.3.3 GetLastError

The **GetLastError** function returns the calling thread's last-error code value. The last-error code is maintained on a per-thread basis. Multiple threads do not overwrite each other's last-error code.

```
DWORD GetLastError  
(  
    VOID  
);
```

#### Function parameter description:

This function has no parameters.

#### Possible function return values

The return value is the calling thread's last-error code value.

#### NOTES:

You should call the **GetLastError** function immediately when a return value of a function indicates that such a call will return useful data. Reason: Some functions call **SetLastError(0)** when they succeed, wiping out the error code set by the most recently failed function.

Most functions in the Win32 API that set the thread's last error code value set it when they fail; a few functions set it when they succeed. Function failure is typically indicated by a return value error code such as **FALSE**, **NULL**, **0xFFFFFFFF**, or **-1**. Some functions call **SetLastError** under conditions of success; these cases are noted in the reference page of each function.

The PROFIBUS driver functions only set the last error code when they fail.

Error codes are 32-bit values (bit 31 is the most significant bit). Bit 29 is called the "Customer code flag" and is reserved for application-defined error codes; no system error code has this bit set. This bit set to one indicates that the error code is defined by the PROFIBUS software. The lower two bytes include the PROFIBUS error code.

To obtain an error string for operating system error codes, use the **FormatMessage** function. A complete list of system error codes can be found in the **WINERROR.H** header file in the Win32 SDK. The list of application-defined error codes can be found in this manual.

## Example

```
#include <windows.h>
#include "pb_err.h"

#define CUSTOMER_CODE_FLAG 0x20000000
...
{
    ULONG ErrorCode;

    // do PROFIBUS IO
    ...

    // get last error
    ErrorCode = GetLastError ();

    // check for Customer code flag
    if (ErrorCode & CUSTOMER_CODE_FLAG)
    {
        // Customer code flag is set: do profibus error handling
        // profibus error code is the low word of ErrorCode
        switch (LOWORD(ErrorCode))
        {
            case E_IF_FATAL_ERROR:
                ...
                break;

            case E_IF_SERVICE_CONSTRAINT_CONFLICT:
                ...
                break;

            ...
        }
    }
    else // System error
    {
        ...
    }
}
```

#### 4.3.4 DeviceIoControl

The **DeviceIoControl** function sends a control code directly to a specified device driver, causing the corresponding logical device to perform the specified operation.

BOOL DeviceIoControl

```
(
    HANDLE          hDevice,
    DWORD           dwIoControlCode,
    LPVOID          lpInBuffer,
    DWORD           nInBufferSize,
    LPVOID          lpOutBuffer,
    DWORD           nOutBufferSize,
    LPDWORD          lpBytesReturned,
    LPOVERLAPPED    lpOverlapped
);
```

##### Function parameter description:

hDevice:	Handle to the device that is to perform the operation. Call the <b>CreateFile</b> function to obtain a device handle.
dwIoControlCode:	Specifies the control code for the operation. This value identifies the specific operation to be performed and the type of device on which the operation is to be performed. The values defined are described later in this section.
lpInBuffer:	Pointer to a buffer that contains the data required to perform the operation. This parameter can be NULL if the dwIoControlCode parameter specifies an operation that does not require input data.
nInBufferSize:	Specifies the size, in bytes, of the buffer pointed to by lpInBuffer.
lpOutBuffer:	Pointer to a buffer that receives the operation's output data. This parameter can be NULL if the dwIoControlCode parameter specifies an operation that does not produce output data.
nOutBufferSize:	Specifies the size, in bytes, of the buffer pointed to by lpOutBuffer.
lpBytesReturned:	Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by lpOutBuffer. lpBytesReturned cannot be NULL. Even when an operation does not produce output data, and lpOutBuffer can be NULL, the <b>DeviceIoControl</b> function makes use of the variable pointed to by lpBytesReturned. After such an operation, the value of the variable is inapplicable.
lpOverlapped:	<p>Pointer to an <b>OVERLAPPED</b> structure.</p> <p>If hDevice was opened with the FILE_FLAG_OVERLAPPED flag, this parameter must point to a valid <b>OVERLAPPED</b> structure. In this case, <b>DeviceIoControl</b> is performed as an overlapped (asynchronous) operation. If the device was opened with FILE_FLAG_OVERLAPPED and lpOverlapped is NULL, the function fails in unpredictable ways.</p> <p>If hDevice was opened without specifying the FILE_FLAG_OVERLAPPED flag, this parameter is ignored, and the <b>DeviceIoControl</b> function does not return until the operation has been completed or an error occurs.</p>

## Possible function return values

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

## NOTES:

If *hDevice* was opened with **FILE\_FLAG\_OVERLAPPED** and the *lpOverlapped* parameter points to an **OVERLAPPED** structure, **DeviceIoControl** is performed as an overlapped (asynchronous) operation. In this case, the **OVERLAPPED** structure must contain a handle to a manual-reset event object created by a call to the **CreateEvent** function.

If the overlapped operation cannot be completed immediately, the function returns **FALSE**, and **GetLastError** returns **ERROR\_IO\_PENDING**, indicating that the operation is executing in the background. When this happens, the operating system sets the event object in the **OVERLAPPED** structure to the non-signaled state before **DeviceIoControl** returns. The system then sets the event object to the signaled state when the operation has been completed. The calling thread can use any of the wait functions to wait for the event object to be signaled, and then use the **GetOverlappedResult** function to determine the results of the operation. The **GetOverlappedResult** function reports the success or failure of the operation and the number of bytes returned in the *lpOutBuffer* buffer.

## Control codes

The following control codes are supported by the PROFIBUS device drivers and defined in the file "pb\_ntdrv.h":

Value	Meaning
IOCTL_PROFI_SET_TIMEOUT	Set time-out values for read and write
	<i>lpInBuffer</i> : Pointer to a <b>PROFI_TIMEOUT</b> data <i>nInBufferSize</i> : Size of <b>PROFI_TIMEOUT</b> data <i>lpOutBuffer</i> : NULL <i>nOutBufferSize</i> : 0
	<i>Devices</i> : General service device if the low-level devices are used: Basic management device if the management or service-oriented high-level Devices are used
	<i>Required Access</i> : Read

IOCTL_PROFI_GET_TIMEOUT	Read time-out values for read and write
<i>lpInBuffer:</i>	NULL
<i>nInBufferSize:</i>	0
<i>lpOutBuffer</i>	Pointer to PROFI_TIMEOUT data
<i>nOutBufferSize</i>	Size of PROFI_TIMEOUT data
Devices:	General service device if the low-level devices are used Basic management device, if the management or service-oriented high-level devices are used
Required Access:	None
IOCTL_PROFI_SET_QUEUE_SIZE	Set the maximum number of frames to be buffered in high-level service-oriented devices
<i>lpInBuffer:</i>	Pointer to integer of type ULONG
<i>nInBufferSize:</i>	Size of ULONG
<i>lpOutBuffer</i>	NULL
<i>nOutBufferSize</i>	0
Devices:	Basic management device
Required Access:	None
IOCTL_PROFI_GET_QUEUE_SIZE	Read the maximum number of frames to be buffered in high-level service-oriented devices. The default queue size is 32.
<i>lpInBuffer:</i>	NULL
<i>nInBufferSize:</i>	0
<i>lpOutBuffer</i>	Pointer to integer of type ULONG
<i>nOutBufferSize</i>	Size of ULONG
Devices:	Basic management device
Required Access:	None
IOCTL_PROFI_GET_OVERRUN_COUNT	Read and reset the overrun count which counts the total number of dismissed frames received from the firmware.
<i>lpInBuffer:</i>	NULL
<i>nInBufferSize:</i>	0
<i>lpOutBuffer</i>	Pointer to integer of type ULONG
<i>nOutBufferSize</i>	size of ULONG
Devices:	Basic management device
Required Access:	None

IOCTL\_PROFI\_CREATE\_DP\_MANAGEMENT  
 IOCTL\_PROFI\_CREATE\_FMS\_MANAGEMENT  
 IOCTL\_PROFI\_CREATE\_FDL\_MANAGEMENT  
 IOCTL\_PROFI\_CREATE\_DP\_SERVICE  
 IOCTL\_PROFI\_CREATE\_DP\_SLAVE\_DATA  
 IOCTL\_PROFI\_CREATE\_DP\_MSAC  
 IOCTL\_PROFI\_CREATE\_FMS\_CR  
 IOCTL\_PROFI\_CREATE\_FDL\_SAP

To minimize the number of existing logical devices to the needs of your application, many logical devices are created on request. In addition, the IoControl functions return the device name of the newly created (or already existing) logical device. The management devices are created by default at system startup time. The control codes are included for completeness and for retrieving the names of the logical devices

lpInBuffer:	Pointer to integer type of ULONG Contains the index of the device to create (CR, DP slave address, SAP)
nInBufferSize:	size of ULONG
lpOutBuffer:	Pointer to an array of characters Contains the name of the created device after successful completion
nOutBufferSize:	Size of array of characters
Devices:	Basic management device DP management device (IOCTL_PROFI_CREATE_DP) FMS management device (IOCTL_PROFI_CREATE_FMS_CR) FDL management device (IOCTL_PROFI_CREATE_FDL_SAP)
Required Access:	None

IOCTL_PROFI_READ_MULTI	<p>Read the first frame received on multiple devices.</p> <p>lpInBuffer: Pointer to an array of handles of the devices to listen for a frame</p> <p>nInBufferSize: Size of the handle array in bytes</p> <p>lpOutBuffer: Pointer to the buffer that receives the frame data</p> <p>nOutBufferSize: Size of the frame buffer The maximum frame size is defined in MAX_FMS_PDU_LENGTH</p> <p>lpBytesReturned: Size of the received frame. If the size of the frame is 0, no frame was received during the time-out time.</p> <p>Devices: All management and frame-oriented high-level devices</p> <p>Required Access: None All devices of the handle array must be opened with read access.</p>
IOCTL_PROFI_GET_DATA_IMAGE	<p>Read a data image from the controller.</p> <p>lpInBuffer: Pointer to PROFI_DATA_IMAGE_DESCR data</p> <p>nInBufferSize: Size of the PROFI_DATA_IMAGE_DESCR data</p> <p>lpOutBuffer: Buffer to receive the data image</p> <p>nOutBufferSize: Size of the data image buffer</p> <p>lpBytesReturned: Size of the read data image</p> <p>Devices: General board device</p> <p>Required Access: None</p>
IOCTL_PROFI_SET_DPS_DATA	<p>Write DP-Slave input data and read DP-Slave input data state.</p> <p>lpInBuffer: DP-Slave input data</p> <p>nInBufferSize: Size of the DP-Slave input data</p> <p>lpOutBuffer: Buffer to receive the input data state</p> <p>nOutBufferSize: sizeof(USIGN8)</p> <p>lpBytesReturned: Size of the read data</p> <p>Devices: DP-Slave input data device</p> <p>Required Access: None</p>

IOCTL_PROFI_SET_DP_BITS	Set Bits in DP slave I/O data image.
lpInBuffer:	Pointer to PROFI_DP_BIT_ACCESS data
nInBufferSize:	Size of PROFI_DP_BIT_ACCESS data
lpOutBuffer:	NULL
nOutBufferSize:	0
lpBytesReturned:	Pointer to variable to receive output byte count, always zero
Devices:	General DP-Master data device
Required Access:	None
IOCTL_PROFI_CLEAR_DP_BITS	Clear Bits in DP slave I/O data image.
lpInBuffer:	Pointer to PROFI_DP_BIT_ACCESS data
nInBufferSize:	Size of PROFI_DP_BIT_ACCESS data
lpOutBuffer:	NULL
nOutBufferSize:	0
lpBytesReturned:	Pointer to variable to receive output byte count, always zero
Devices:	General DP-Master data device
Required Access:	None



## Example

```
{
    HANDLE hBasicMgmt;
    HANDLE hCR[5];
    ULONG   cr, Bytes;
    char    deviceName[100];
    BYTE    dataRead[MAX_FMS_PDU_LENGTH];

    // Open basic management device
    hBasicMgmt = CreateFile (("\\\\.\\PROFIBUS\\Board0\\Pb0\\Management",
                                GENERIC_READ,0,NULL,OPEN_EXISTING,
                                FILE_ATTRIBUTE_NORMAL,NULL));
    if (hBasicMgmt != INVALID_HANDLE_VALUE)
    {
        for (cr = 0; cr < 5; cr++)
        {
            // create the FMS CR device
            if (DeviceIoControl(hBasicMgmt, (ULONG) IOCTL_PROFI_CREATE_FMS_CR, &cr,
                                sizeof(ULONG), deviceName, 100, &nBytes, NULL))
            {
                // open the FMS CR device
                hCR[cr] = CreateFile (deviceName, GENERIC_READ | GENERIC_WRITE, 0, NULL,
                                        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
                if (hCR[cr] == INVALID_HANDLE_VALUE)
                {
                    // do error handling - open of device failed
                }
                {
                    else
                    {
                        ... // do error handling - creation of device failed
                    }
                }
            }
            // all CR devices open
            ...
            // read on all CR devices
            if( DeviceIoControl(hCR, (ULONG) IOCTL_PROFI_READ_MULTI, (LPVOID) hCR,
                                5 * sizeof(HANDLE), dataRead, MAX_FMS_PDU_LENGTH,
                                &nBytes, NULL))
            {
                if (nBytes > 0)
                {
                    // Frame received
                }
                else
                {
                    // No frame received during time-out
                }
                ...
            }
        }
        ...
    }
}
```

## 4.3.5 ReadFile

The **ReadFile** function reads data from a device. See the description of each logical device for a description of the data to read. File positions are considered only during read operations from the general data devices.

BOOL ReadFile

```
(
HANDLE          hFile,
LPVOID          lpBuffer,
DWORD           nNumberOfBytesToRead,
LPDWORD         lpNumberOfBytesRead,
LPOVERLAPPED    lpOverlapped
);
```

### Function parameter description:

hFile:	Identifies the file to be read. The file handle must have been created with GENERIC_READ access to the file. For asynchronous read operations, hFile can be any handle opened with the FILE_FLAG_OVERLAPPED flag by the <b>CreateFile</b> function.
lpBuffer:	Points to the buffer that receives the data read from the device.
nNumberOfBytesToRead:	Specifies the maximum number of bytes to be read from the device.
lpNumberOfBytesRead:	Points to the number of bytes read. If lpOverlapped is NULL, lpNumberOfBytesRead cannot be NULL. If lpOverlapped is not NULL, lpNumberOfBytesRead can be NULL. If this is an overlapped read operation, the number of bytes read can be fetched by calling <b>GetOverlappedResult</b> .
lpOverlapped:	Points to an <b>OVERLAPPED</b> structure. This structure is required if hFile was created with FILE_FLAG_OVERLAPPED. If hFile was opened with FILE_FLAG_OVERLAPPED, the lpOverlapped parameter must not be NULL. It must point to a valid <b>OVERLAPPED</b> structure. If hFile was created with FILE_FLAG_OVERLAPPED and lpOverlapped is NULL, the function can incorrectly report that the read operation is complete. If hFile was opened with FILE_FLAG_OVERLAPPED and lpOverlapped is not NULL, the read operation starts at the offset specified in the <b>OVERLAPPED</b> structure and <b>ReadFile</b> may return before the read operation has been completed. In this case, <b>ReadFile</b> returns FALSE, and the <b>GetLastError</b> function returns ERROR_IO_PENDING. This allows the calling process to continue while the read operation finishes. The event specified in the <b>OVERLAPPED</b> structure is set to the signaled state upon completion of the read operation. If hFile was not opened with FILE_FLAG_OVERLAPPED and lpOverlapped is NULL, the read operation starts at the current file position, and <b>ReadFile</b> does not return until the operation has been completed. If hFile is not opened with FILE_FLAG_OVERLAPPED and lpOverlapped is not NULL, the read operation starts at the offset specified in the <b>OVERLAPPED</b> structure. <b>ReadFile</b> does not return until the read operation has been completed.

### Possible function return values:

- If the function succeeds, the return value is **TRUE**. If the return value is TRUE and the number of bytes read is zero, no data are available at the device.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

### NOTES:

Applications must neither read from nor write to the input buffer that a read operation is using until the read operation completes. A premature access to the input buffer may lead to corruption of the data read into that buffer.

The `ReadFile` function may fail and return `ERROR_INVALID_USER_BUFFER` or `ERROR_NOT_ENOUGH_MEMORY` whenever there are too many outstanding asynchronous I/O requests.

If you read from the general service device, the `comm_ref` field of the service descriptor block contains a number of status bits in the high byte. Only the low byte contains the communication reference of the service.

## Usage

<b>Board device:</b>	Reads the version information of the PROFIBUS protocol firmware	
	<code>lpBuffer:</code>	pointer to an array of characters
	<code>nNumberOfBytesToRead:</code>	the maximum size of the version information is defined in <code>VERSION_STRING_LENGTH</code>
<b>Service-oriented devices:</b>	Reads a received frame. The frame consists of the service description followed by the service and primitive specific data.	
	<code>lpBuffer:</code>	pointer to the buffer that receives the frame data
	<code>nNumberOfBytesToRead:</code>	size of the buffer pointed by <code>lpBuffer</code> . A frame could have the maximum size defined in <code>MAX_FMS_PDU_LENGTH</code>
<b>Data-oriented devices:</b>	<code>lpNumberOfBytesRead:</code>	size of the received frame. If the size of the frame is 0 and the function succeeded, no frame was received during the time-out time.
	Reads DP data. The DP slave data devices check the status information of the slave and return the error <code>E_SLAVE_ERROR</code> if the slave status is bad.	
	<code>lpBuffer:</code>	Pointer to the buffer that receives the DP data
	<code>nNumberOfBytesToRead</code>	Size of the buffer. The maximum amount of bytes to read on DP slave data devices is the maximum read size of the slave.

## Examples

```
{  
    HANDLE hBoard;                                // Handle of the board device  
    HANDLE hService;                             // Handle of the general service device  
    char    firmwareVersion[VERSION_STRING_LENGTH];  
  
    // Open board and service device  
    ...  
  
    // Read the firmware version info from the board device  
    if(ReadFile(hBoard,dataVersion,VERSION_STRING_LENGTH,&nBytes,NULL))  
    {  
        // read version info  
        ...  
    }  
  
    // Read from the service device  
    if(ReadFile(hService,dataService,MAX_FMS_PDU_LENGTH,&nBytes,NULL))  
    {  
        if (nBytes > 0)  
            // Frame received  
        else  
            // No frame received during time-out  
  
        ...  
    }  
}
```

```
{  
    HANDLE hSlave3;                                // Handle of the DP slave data device of the slave 3  
    BYTE    dataService[MAX_FMS_PDU_LENGTH];  
  
    // Create and open the DP slave data device  
    ...  
  
    // Read form the DP slave data device of the slave 3  
    if(ReadFile(hSlave3,(LPVOID) &dataSlave3,sizeof(dataSlave3),&nBytes,NULL))  
    {  
        // read DP data  
        ...  
    }  
}
```

### 4.3.6 ReadFileEx

The **ReadFileEx** function reads data from a file asynchronously. It is designed solely for asynchronous operation, unlike the **ReadFile** function, which is designed for both synchronous and asynchronous operation. **ReadFileEx** lets an application perform other processing during a file read operation.

The **ReadFileEx** function reports its completion status asynchronously, calling a specified completion routine when reading is completed and the calling thread is in an alertable wait state.

BOOL ReadFileEx

```
(
    HANDLE                hFile,
    LPVOID                lpBuffer,
    DWORD                nNumberOfBytesToRead,
    LPOVERLAPPED          lpOverlapped,
    LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

#### Function parameter description:

hFile:	An open handle that specifies the device to be read from. This file handle must have been created with the FILE_FLAG_OVERLAPPED flag and must have GENERIC_READ access to the file.
lpBuffer:	Points to a buffer that receives the data read from the file. This buffer must remain valid for the duration of the read operation. The application should not use this buffer until the read operation is completed.
nNumberOfBytesToRead:	Specifies the maximum number of bytes to be read from the file. If nNumberOfBytesToRead is zero, this function does nothing.
lpOverlapped:	Points to an <b>OVERLAPPED</b> data structure that supplies data to be used during the asynchronous (overlapped) file read operation. If the device specified by hFile supports the concept of byte offsets (these are the general data devices, e.g. "\\PROFIBUS\\Board0\\DpData") , the caller of <b>ReadFileEx</b> must specify a byte offset at which reading should begin. The caller specifies the byte offset by setting the <b>OVERLAPPED</b> structure's Offset member; the OffsetHigh member must be set to 0. If the file entity specified by hFile does not support the concept of byte, the caller must set the Offset and OffsetHigh members to zero, or <b>ReadFileEx</b> fails. The <b>ReadFileEx</b> function ignores the <b>OVERLAPPED</b> structure's hEvent member. An application is free to use that member for its own purposes in the context of a <b>ReadFileEx</b> call. <b>ReadFileEx</b> signals completion of its read operation by calling, or queuing a call to, the completion routine pointed to by lpCompletionRoutine, so it does not need an event handle. The <b>ReadFileEx</b> function does use the <b>OVERLAPPED</b> structure's Internal and InternalHigh members. An application should not set these members. The <b>OVERLAPPED</b> data structure pointed to by lpOverlapped must remain valid for the duration of the read operation. It should not be a variable that can go out of scope while the file read operation is in progress.
lpCompletionRoutine:	Points to the completion routine to be called when the read operation is complete and the calling thread is in an alertable wait state. For more information about the completion routine, see <b>FileIOCompletionRoutine</b> .

**Possible function return values**(defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call GetLastError.

If the function succeeds, the calling thread has an asynchronous I/O operation pending: the overlapped read operation from the file. When this I/O operation completes and the calling thread is blocked in an alertable wait state, the system calls the function pointed to by *lpCompletionRoutine*, and the wait state completes with a return code of **WAIT\_IO\_COMPLETION**.

If the function succeeds and the file reading operation completes but the calling thread is not in an alertable wait state, the system queues the completion routine call, holding the call until the calling thread enters an alertable wait state.

## NOTES:

Applications must neither read from nor write to the input buffer that a read operation is using until the read operation completes. A premature access to the input buffer may lead to corruption of the data read into that buffer.

The **ReadFileEx** function may fail if there are too many outstanding asynchronous I/O requests. In the event of such a failure, **GetLastError** can return **ERROR\_INVALID\_USER\_BUFFER** or **ERROR\_NOT\_ENOUGH\_MEMORY**.

If *hFile* is a handle to a named pipe or other file entity that does not support the byte-offset concept, the **Offset** and **OffsetHigh** members of the **OVERLAPPED** structure pointed to by *lpOverlapped* must be zero, or **ReadFileEx** fails.

An application uses the **WaitForSingleObjectEx**, **WaitForMultipleObjectsEx**, and **SleepEx** functions to enter an alertable wait state.

## Usage

There is no sense in using the **ReadFileEx** function with board or data-oriented devices, because these system calls are served immediately by the PROFIBUS device drivers. A read operation may become pending only on the service-oriented devices.

<b>Service-oriented devices:</b>	Starts the asynchronous read of a received frame	
	<i>lpBuffer</i> :	Pointer to the buffer that receives the frame data
	<i>nNumberOfBytesToRead</i> :	The maximum size of a frame is defined in <b>MAX_FMS_PDU_LENGTH</b> .

### 4.3.7 WriteFile

The **WriteFile** function writes data to a file and is designed for both synchronous and asynchronous operation.

```

BOOL WriteFile
(
    HANDLE          hFile,
    LPCVOID         lpBuffer,
    DWORD           nNumberOfBytesToWrite,
    LPDWORD         lpNumberOfBytesWritten,
    LPOVERLAPPED    lpOverlapped
);

```

#### Function parameter description:

hFile:	Identifies the file to be written to. The file handle must have been created with <b>GENERIC_WRITE</b> access to the file. For asynchronous write operations, hFile can be any handle opened with the <b>FILE_FLAG_OVERLAPPED</b> flag by the <b>CreateFile</b> function.
lpBuffer:	Points to the buffer containing the data to be written to the file.
nNumberOfBytesToWrite:	Specifies the number of bytes to write to the file.
lpNumberOfBytesWritten:	Points to the number of bytes written by this function call. If lpOverlapped is NULL, lpNumberOfBytesWritten cannot be NULL. If lpOverlapped is not NULL, lpNumberOfBytesWritten can be NULL. If this is an overlapped write operation, the number of bytes written can be fetched by calling <b>GetOverlappedResult</b> .
lpOverlapped:	<p>Points to an <b>OVERLAPPED</b> structure. This structure is required if hFile was opened with <b>FILE_FLAG_OVERLAPPED</b>.</p> <p>If hFile was opened with <b>FILE_FLAG_OVERLAPPED</b>, the lpOverlapped parameter must not be NULL. It must point to a valid <b>OVERLAPPED</b> structure. If hFile was opened with <b>FILE_FLAG_OVERLAPPED</b> and lpOverlapped is NULL, the function can incorrectly report that the write operation is complete.</p> <p>If hFile was opened with <b>FILE_FLAG_OVERLAPPED</b> and lpOverlapped is not NULL, the write operation starts at the offset specified in the <b>OVERLAPPED</b> structure, and <b>WriteFile</b> may return before the write operation has been completed. In this case, <b>WriteFile</b> returns <b>FALSE</b>, and the <b>GetLastError</b> function returns <b>ERROR_IO_PENDING</b>. This allows the calling process to continue processing while the write operation is being completed. The event specified in the <b>OVERLAPPED</b> structure is set to the signaled state upon completion of the write operation.</p> <p>If hFile was not opened with <b>FILE_FLAG_OVERLAPPED</b> and lpOverlapped is NULL, the write operation starts at the current file position, and <b>WriteFile</b> does not return until the operation has been completed.</p> <p>If hFile was not opened with <b>FILE_FLAG_OVERLAPPED</b> and lpOverlapped is not NULL, the write operation starts at the offset specified in the <b>OVERLAPPED</b> structure, and <b>WriteFile</b> does not return until the write operation has been completed.</p>

**Possible function return values**(defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

## NOTES:

Applications must neither read from nor write to the output buffer that a write operation is using until the write operation completes. Premature access of the output buffer may lead to corruption of the data written from that buffer.

The WriteFile function may fail with **ERROR\_INVALID\_USER\_BUFFER** or **ERROR\_NOT\_ENOUGH\_MEMORY** whenever there are too many outstanding asynchronous I/O requests.

## Usage

<b>Service-oriented devices:</b>	<p>Send a frame. The frame consists of the service description followed by the service and primitive-specific data.</p> <p>lpBuffer: Pointer to the frame to send</p> <p>nNumberOfBytesToWrite: Size of the frame</p> <p>The WriteFile function fails with the PROFIBUS error E_IF_NO_CNTRL_RES if the time-out for sending of the frame elapsed.</p>
<b>Data-oriented devices:</b>	<p>Writes DP data. The DP slave data devices check the status information of the slave and return the error <b>E_SLAVE_ERROR</b> if the slave status is bad.</p> <p>lpBuffer: Pointer to the DP data which should be written.</p> <p>nNumberOfBytesToWrite: Size of the DP data to write. The maximum amount of bytes to write to DP slave data devices is the maximum write size of the slave.</p>



## Example

```
{
    HANDLE          hService;                // Handle of the general service device
    HANDLE          hSlave3;    // Handle of the DP slave data device of the slave 3
    T_PROFI_SERVICE_DESCR sdb;                // Service description
    T_VAR_READ_REQ   readReq;                // Read request
    BYTE            dataService[MAX_FMS_PDU_LENGTH];
    USIGN8          invokeId

    ...

    // Open devices and create the DP slave data device
    ...

    // All devices open with read access

    // Send a FMS_READ request

    // Fill the service description
    sdb.layer        = FMS;
    sdb.service      = FMS_READ;
    sdb.primitive    = REQ;
    sdb.invoke_id    = ++invokeId;
    sdb.comm_ref     = 5;

    // Fill the read request
    readReq.acc_spec.tag      = ACCESS_INDEX;
    readReq.acc_spec.id.index = 20;
    readReq.subindex         = 0;

    // Create the frame
    memcpy(dataService,&sdb,sizeof(T_PROFI_SERVICE_DESCR));
    memcpy(dataService + sizeof(T_PROFI_SERVICE_DESCR),&readReq, sizeof(T_VAR_READ_REQ));

    // Send the frame
    if(!WriteFile(hService,dataService,sizeof(T_PROFI_SERVICE_DESCR) +
        sizeof(T_VAR_READ_REQ),&nBytes,NULL))
    {
        // error handling
        ...
    }

    // write data to the DP slave data device of the slave 3
    if(!WriteFile(hSlave3,(LPVOID)&dataSlave3,sizeof(dataSlave3),&nBytes,NULL))
    {
        // error handling
        ...
    }
}
```

## 4.3.8 WriteFileEx

The **WriteFileEx** function writes data to a file. It is designed solely for asynchronous operation, unlike **WriteFile**, which is designed for both synchronous and asynchronous operation. **WriteFileEx** reports its completion status asynchronously, calling a specified completion routine when writing is completed and the calling thread is in an alertable wait state.

BOOL WriteFileEx

```
(
HANDLE                                hFile,
LPCVOID                              lpBuffer,
DWORD                                nNumberOfBytesToWrite,
LPOVERLAPPED                         lpOverlapped,
LPOVERLAPPED_COMPLETION_ROUTINE     lpCompletionRoutine
);
```

### Function parameter description:

hFile:	An open handle that specifies the device to be written to. This file handle must have been created with the FILE_FLAG_OVERLAPPED flag and with GENERIC_WRITE access to the file.
lpBuffer:	Points to the buffer containing the data to be written to the file. This buffer must remain valid for the duration of the write operation. The caller must not use this buffer until the write operation is completed.
nNumberOfBytesToWrite:	Specifies the number of bytes to write to the file. If nNumberOfBytesToWrite is zero, this function does nothing.
lpOverlapped:	Points to an <b>OVERLAPPED</b> data structure that supplies data to be used during the overlapped (asynchronous) write operation. For devices that support byte offsets (these are the general data devices, e.g. "\\PROFIBUS\\Board0\\Pb0\\DpData"), you must specify a byte offset at which to start writing to the file. Specify this offset by setting the Offset member of the <b>OVERLAPPED</b> structure and setting OffsetHigh to zero. For files that do not support byte offsets, set Offset and OffsetHigh to zero, or <b>WriteFileEx</b> fails. The <b>WriteFileEx</b> function ignores the <b>OVERLAPPED</b> structure's hEvent member. An application is free to use that member for its own purposes in the context of a <b>WriteFileEx</b> call. <b>WriteFileEx</b> signals completion of its writing operation by calling, or queuing a call to, the completion routine pointed to by <b>lpCompletionRoutine</b> , so it does not need an event handle. The <b>WriteFileEx</b> function uses the Internal and InternalHigh members of the <b>OVERLAPPED</b> structure. Do not change the value of these members. The <b>OVERLAPPED</b> data structure must remain valid for the duration of the write operation. It should not be a variable that can go out of scope while the write operation is pending completion.
lpCompletionRoutine:	Points to a completion routine to be called when the write operation has been completed and the calling thread is in an alertable wait state. For more information about this completion routine, see <b>FileIOCompletionRoutine</b> .

**Possible function return values**(defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

If the WriteFileEx function succeeds, the calling thread has an asynchronous I/O operation pending: the overlapped write operation to the device. When this I/O operation finishes and the calling thread is blocked in an alertable wait state, the operating system calls the function pointed to by lpCompletionRoutine, and the wait completes with a return code of WAIT\_IO\_COMPLETION.

If the function succeeds and the file-writing operation finishes but the calling thread is not in an alertable wait state, the system queues the call to \*lpCompletionRoutine, holding the call until the calling thread enters an alertable wait state.

### NOTES:

Applications must neither read from nor write to the output buffer that a write operation is using until the write operation completes. Premature access of the output buffer may lead to corruption of the data written from that buffer.

The WriteFileEx function may fail, returning the messages ERROR\_INVALID\_USER\_BUFFER or ERROR\_NOT\_ENOUGH\_MEMORY if there are too many outstanding asynchronous I/O requests.

An application uses the WaitForSingleObjectEx, WaitForMultipleObjectsEx, MsgWaitForMultipleObjectsEx, SignalObjectAndWait, and SleepEx functions to enter an alertable wait state.

### Usage

There is no sense in using the **WriteFileEx** function with board or data-oriented devices, because this system calls are served immediately by the PROFIBUS device drivers. Only on the service-oriented devices a write operation may become pending.

<b>Service-oriented devices:</b>	Starts the asynchronous send of a frame	
	lpBuffer.	Pointer to the frame to send
	nNumberOfBytesToWrite:	Size of the frame

### 4.3.9 GetOverlappedResult

The **GetOverlappedResult** function returns the results of an overlapped operation on the specified file, named pipe, or communications device.

```

BOOL GetOverlappedResult
(
    HANDLE                hFile,
    LPOVERLAPPED          lpOverlapped,
    LPDWORD                lpNumberOfBytesTransferred,
    BOOL                   bWait
);

```

#### Function parameter description:

hFile:	Identifies the device. This is the same handle that was specified when the overlapped operation was started by a call to the ReadFile, WriteFile, or DeviceIoControl function.
lpOverlapped:	Points to an OVERLAPPED structure that was specified when the overlapped operation was started.
lpNumberOfBytesTransferred:	This value is undefined. Points to a 32-bit variable that receives the number of bytes that were actually transferred by a read or write operation.
bWait:	Specifies whether the function should wait for the pending overlapped operation to be completed. If TRUE, the function does not return until the operation has been completed. If FALSE and the operation is still pending, the function returns FALSE and the <b>GetLastError</b> function returns ERROR_IO_INCOMPLETE.

#### Possible function return values:

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

#### NOTES:

The results reported by the **GetOverlappedResult** function are those of the specified handle's last overlapped operation to which the specified OVERLAPPED structure was provided and for which the operation's results were pending. A pending operation is indicated when the function that started the operation returns FALSE and the **GetLastError** function returns ERROR\_IO\_PENDING. When an I/O operation is pending, the function that started the operation resets the hEvent member of the OVERLAPPED structure to the non-signaled state. Then when the pending operation has been completed, the system sets the event object to the signaled state.

If the bWait parameter is TRUE, **GetOverlappedResult** determines whether the pending operation has been completed by waiting for the event object to be in the signaled state.

If the `hEvent` member of the `OVERLAPPED` structure is `NULL`, the system uses the state of the `hFile` handle to signal when the operation has been completed. Use of file, named pipe, or communications-device handles for this purpose is discouraged. It is safer to use an event object because of the confusion that can occur when multiple simultaneous overlapped operations are performed on the same file, named pipe, or communications device. In this situation, there is no way to know which operation caused the object's state to be signaled.

Specify a manual-reset event object in the `OVERLAPPED` structure. If an auto-reset event object is used, the event handle must not be specified in any other wait operation in the interval between starting the overlapped operation and the call to `GetOverlappedResult`. For example, the event object is sometimes specified in one of the wait functions to wait for the operation's completion. When the wait function returns, the system sets an auto-reset event's state to non-signaled, and a subsequent call to `GetOverlappedResult` with the `bWait` parameter set to `TRUE` causes the function to be blocked indefinitely.

## 4.3.10 SetFilePointer

The only type of PROFIBUS devices that support the concept of file pointers is the general data device. The **SetFilePointer** function moves the file pointer of an open general data device.

DWORD SetFilePointer

```
(
    HANDLE      hFile,
    LONG        IDistanceToMove,
    PLONG       lpDistanceToMoveHigh,
    DWORD       dwMoveMethod
);
```

### Function parameter description:

hFile:	Identifies the file whose file pointer is to be moved. The file handle must have been created with GENERIC_READ or GENERIC_WRITE access to the file.
IDistanceToMove:	Specifies the number of bytes to move the file pointer. A positive value moves the pointer forward in the file and a negative value moves it backward.
lpDistanceToMoveHigh:	Must be NULL for PROFIBUS devices
dwMoveMethod:	Specifies the starting point for the file pointer move. This parameter can be one of the following values:
<b>Value</b>	<b>Meaning</b>
FILE_BEGIN	The starting point is zero or the beginning of the file. If FILE_BEGIN is specified, DistanceToMove is interpreted as an unsigned location for the new file pointer.
FILE_CURRENT	The current value of the file pointer is the starting point.
FILE_END	Cannot be used for PROFIBUS devices.

### Possible function return values:

- If the SetFilePointer function succeeds, the return value is the low-order doubleword of the new file pointer.
- If the function fails, the return value is 0xFFFFFFFF. To obtain extended error information, call **GetLastError**.

### NOTES:

You should be careful when setting the file pointer in a multithreaded application. For example, an application whose threads share a file handle, update the file pointer, and read from the file must protect this sequence by using a critical section object or mutex object.

The PROFIBUS general data device doesn't change the position of the file pointer with a read or write operation. The file pointer is only changed with the SetFilePointer function. You do not have to set the file pointer before every ReadFile or WriteFile call. Once set, the file pointer stays at the position.

### Example

```
{  
    HANDLE hData;                                // Handle of the general data device  
    DWORD  filePointer                           // File pointer  
    LONG   distance;                             // Distance to move  
    ...  
  
    // Open general data device  
    ...  
  
    filePointer = SetFilePointer (hData, distance, NULL, FILE_BEGIN)  
  
    if (filePointer == 0xffffffff)  
    {  
        // error handling  
        ...  
    }  
  
    // Continue with read and write to the general data device  
    ...  
}
```

## 4.3.11 FileIOCompletionRoutine

The **FileIOCompletionRoutine** function is called when an asynchronous I/O function (**ReadFileEx** or **WriteFileEx**) is completed and the calling thread is in an alertable wait (using the **SleepEx**, **WaitForSingleObjectEx**, or **WaitForMultipleObjectsEx** function with the *fAlertable* flag set to TRUE).

VOID FileIOCompletionRoutine

```
(
    DWORD          dwErrorCode,
    DWORD          dwNumberOfBytesTransferred,
    LPOVERLAPPED   lpOverlapped
);
```

### Function parameter description:

**dwErrorCode:** Specifies the I/O completion status. This parameter may be one of the following values:

Value	Meaning
0	The I/O was successful.
ERROR_HANDLE_EOF	The ReadFileEx function tried to read past the end of the file.

**dwNumberOfBytesTransferred:** Specifies the number of bytes transferred. If an error occurs, this parameter is zero.

**lpOverlapped:** Points to the **OVERLAPPED** structure specified by the asynchronous I/O function.

Windows does not use the hEvent member of the **OVERLAPPED** structure; the calling application may use this member to pass information to the completion routine. Windows does not use the **OVERLAPPED** structure after the completion routine is called, so the completion routine can de-allocate the memory used by the overlapped structure.

### Possible function return values:

This function does not return a value.

### NOTES:

The **FileIOCompletionRoutine** function is a placeholder for an application-defined or library-defined function name.

Returning from this function allows another pending I/O completion routine to be called. All waiting completion routines are called before the alertable thread's wait is satisfied with a return code of **WAIT\_IO\_COMPLETION**. Windows may call the waiting completion routines in any order. They may or may not be called in the order the I/O functions are completed.

Each time Windows calls a completion routine, it uses some of the application's stack. If the completion routine does additional asynchronous I/O and alertable waits, the stack may grow.



### Usage

#### FileIOCompletionRoutine for ReadFileEx:

**Service-oriented devices:** Fetches the result of the asynchronous read of a received frame.

dwNumberOfBytesTransferred:	Size of the received frame. If the size of the frame is 0, no frame was received during the time-out time
-----------------------------	-----------------------------------------------------------------------------------------------------------

#### FileIOCompletionRoutine for WriteFileEx:

**Service-oriented devices:** Fetches the result of the asynchronous write of a sent frame.

dwNumberOfBytesTransferred:	Size of the sent frame.
-----------------------------	-------------------------

#### **4.4 PROFIBUS APPLICATION PROGRAM INTERFACE**

The PROFIBUS Application Program Interface (PAPI) running in the Windows 2000 or Windows NT environment, provides the same set of functions as the PROFIBUS API running in a Windows ME, Windows 9x, Windows 3.1x or MS-DOS environment.

The PROFIBUS API consists of these functions:

<b>init_profibus</b>	Initialize interface (only board 0)
<b>profi_set_default</b>	Initialize interface
<b>profi_end</b>	Shut down interface
<b>profi_snd_req_res</b>	Send frame
<b>profi_rcv_con_ind</b>	Receive frame
<b>profi_set_data</b>	Write data
<b>profi_get_data</b>	Read data
<b>profi_set_dps_input_data</b>	Write DP-Slave input data
<b>profi_get_dps_input_data</b>	Read DP-Slave input data
<b>profi_get_dps_output_data</b>	Read DP-Slave output data
<b>profi_get_versions</b>	Read version strings
<b>profi_get_serial_device_number</b>	Read serial device number

The service-oriented functions use the general service device of the low-level kernel mode driver and the data-oriented functions use the general DP-Master data device, DP-Slave input data device or DP-Slave output data device of the low-level kernel mode driver. The compatibility mode PAPI interface does not use the functionality provided by the protocol kernel mode driver.

## 4.4.1 Initialization and Shut down

The initialization functions **init\_profibus**, and **profi\_set\_default** are used to initialize the PROFIBUS API and open the low-level devices of the PROFIBUS hardware driver.

### 4.4.1.1 Init-Profibus

The **init\_profibus** function is used to initialize the PROFIBUS API and to open the low-level devices of **interface 0** (board 0) of the PROFIBUS hardware driver. The function has to be called before any other function of PROFIBUS-API is called

```
INT16 init_profibus
(
    IN USIGN32   DprAddress,
    IN USIGN16   IOPortAddress,
    IN PB_BOOL   Dummy
);
```

#### Function parameter description:

DprAddress:	Parameter is without meaning. The DprAddress is set automatically by the PROFIBUS hardware driver.
IOPortAddress:	Parameter is without meaning. The IOPortAddress is set automatically by the PROFIBUS hardware driver.
Dummy:	Parameter is a dummy parameter.

#### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(0x00)	Interface is initialized
- E_IF_CMI_ERROR	(0x14)	Can not set timeout values
- E_IF_SERVICE_NOT_EXECUTABLE	(0x19)	Application has called init_profibus before
- E_IF_READING_REGISTRY	(0xF3)	Error reading registry
- E_IF_OS_ERROR	(0xFF)	Can not open low-level device(s)

**NOTES:**

The `init_profibus` function initializes only board 0 and only polling mode (timeout = 0) is supported.

**Example**

```
...
#include "pb_if.h"
...
{
    INT16 rc;

    if (E_OK == (rc = init_profibus(0,0,0)))
    {
        // Compatibility mode PAPI initialized
        ..
    }
}
```

#### 4.4.1.2 Profi-Set-Default

The **profi\_set\_default** function is used to initialize the PROFIBUS API and to open the low-level devices of the **desired interface** (board) of the PROFIBUS hardware driver. The function has to be called before any other function of PROFIBUS-API is called

```
INT16 profi_set_default
(
    IN USIGN8    Board,
    IN USIGN8    Channel,
    IN USIGN32   ReadTimeout,
    IN USIGN32   WriteTimeout
);
```

##### Function parameter description:

Board:                Number between 0..9 of the board to work on  
Channel:             PROFIBUS channel number (not supported).  
ReadTimeout:        ReceiveTimeout in msec (WAIT\_FOR\_EVER for infinity wait).  
WriteTimeout:        Send Timeout in msec (WAIT\_FOR\_EVER for infinity wait).

##### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(0x00)	Interface is initialized
- E_IF_CMI_ERROR	(0x14)	Can not set timeout values
- E_IF_SERVICE_NOT_EXECUTABLE	(0x19)	Application has called init_profibus before
- E_IF_READING_REGISTRY	(0xF3)	Error reading registry
- E_IF_OS_ERROR	(0xFF)	Can not open low-level device(s)

##### NOTES:

**One process can only use one board. To use more than one board, new process(es) must be started.**

##### Example

```
#include "pb_if.h"
...
{
    INT16    rc;
    USIGN8   boardNr;
    USIGN32  readTimeout, writeTimeout;
    ...
    if (E_OK == (rc = profi_set_default(boardNr,0,readTimeout,writeTimeout)))
    {
        ... // PAPI is initialized
    }
}
```

#### 4.4.1.3 Profi-End

The **profi\_end** function is used to shut down the PROFIBUS API. This means that the low-level devices will be closed.

```
INT16 profi_end
(
    VOID
);
```

**Possible function return values(defined in the header file PB\_ERR.H):**

- E_OK	(0x00)	Shutdown excuted successfully
--------	--------	-------------------------------

#### Example

```
...
#include "pb_if.h"
...
{
    // Initialize PROFIBUS API
    ...

    // Shut down PROFIBUS API
    profi_end();
}
```

## 4.4.2 Send / Receive Interface

The send/receive interface provides by means for both control flow and data flow between host and controller.

**Data flow** between the application and the communication is described by a service invariant and a large number of service specific data structures.

**Control flow** is directed by means of two functions, which control the data flow in both directions.

The two cases described above are covered by two interface functions in the Softing PROFIBUS implementations.

The **profi\_snd\_req\_res** function is used for sending requests and responses. The **profi\_rcv\_con\_ind** function is used to receive confirmations and indications.

### 4.4.2.1 Profi-Snd-Req-Res

The **profi\_snd\_req\_res** function is used for sending requests and responses to PROFIBUS interface.

```
INT16 profi_snd_req_res
(
    IN T_PROFI_SERVICE_DESCR*    pSdb,
    IN VOID*                      pData,
    IN PB_BOOL                    Dummy
);
```

#### Function parameter description:

pSdb:	Pointer to the data structure of type T_PROFI_SERVICE_DESCR
pData:	Pointer to service specific parameters and data
Dummy:	Dummy parameter

#### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(0)	Function executed correctly
- E_IF_FATAL_ERROR	(7)	Unrecoverable error on PROFIBUS controller
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_INVALID_LAYER	(12)	Invalid layer
- E_IF_INVALID_SERVICE	(13)	Invalid service identifier
- E_IF_INVALID_PRIMITIVE	(14)	Invalid service primitive
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_RESOURCE_UNAVAILABLE	(21)	No resource available
- E_IF_NO_PARALLEL_SERVICES	(22)	No parallel services allowed
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service temporarily not executable

- E_IF_SERVICE_NOT_SUPPORTED	(24)	Service not supported in subset
- E_IF_SERVICE_NOT_EXECUTABLE	(25)	Service not executable
- E_IF_INVALID_PARAMETER	(30)	Invalid parameter in REQ or RES
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

## NOTES:

If `profi_snd_req_res` function fails with `E_IF_NO_CTRL_RES`, the controller did not respond during the send time-out value specified in `profi_set_default`. You can obtain extended error information with `GetLastError` if the function returns `E_IF_OS_ERROR`.

## Example

```
...
#include "pb_if.h"
...
{
    T_PROFI_SERVICE_DESCR sdb;                // Service description
    T_VAR_READ_REQ        readReq;            // Read request
    INT16                 rc;
    USIGN8                 invokeId;

    // initialize the PROFIBUS API
    ...

    // send a FMS_READ request
    // fill the service description
    sdb.layer              = FMS;
    sdb.service             = FMS_READ;
    sdb.primitive           = REQ;
    sdb.invoke_id           = ++invokeId;
    sdb.comm_ref            = 5;

    // fill the read request
    readReq.acc_spec.tag     = ACCESS_INDEX;
    readReq.acc_spec.id.index = 20;
    readReq.subindex         = 0;

    if (E_OK != (rc = profi_snd_req_res(&sdb, (void *)&readReq, PB_TRUE)))
    {
        // Error handling
        ...
    }
    ...
}
```



#### 4.4.2.2 Profi-Rcv-Con-Ind

The **profi\_rcv\_con\_ind** function is used to receive a service indication or service confirmation from the PROFIBUS interface when available.

```
INT16 profi_rcv_con_ind
(
    IN      T_PROFI_SERVICE_DESCR* pSdb,
    IN      VOID*                  pData,
    INOUT   USIGN16*               pDataLength
);
```

##### Function parameter description:

pSdb: Buffer for service description block  
 pData: Buffer for service specific data block  
 pDataLen: On function invocation: maximal size of data block  
 On function return: actual size of service specific data block

The function returns **CON\_IND\_RECEIVED** to signal that a confirmation or indication is available.

##### Possible function return values (defined in the header file PB\_ERR.H):

- NO_CON_IND_RECEIVED	(0)	There is no confirmation or indication
- CON_IND_RECEIVED	(1)	Confirmation or indication is available
- E_IF_FATAL_ERROR	(7)	Unrecoverable error on PROFIBUS controller
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_INVALID_DATA_SIZE	(15)	Size of data block provided not sufficient
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_RESOURCE_UNAVAILABLE	(21)	No resource available
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

##### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

## Example

```
...
#include "pb_if.h"
...
{
    T_PROFI_SERVICE_DESCR   sdb;                                // Service description
    BYTE                    data[MAX_FMS_PDU_LENGTH];           // Data buffer
    USIGN16                 dataLen;
    INT16                   rc;

    // Initialize PROFIBUS API
    ...
    // Receive service indication or service confirmation
    dataLen = sizeof(data);
    rc = profi_rcv_con_ind(&sdb, &data, &dataLen);

    if (rc == CON_IND_RECEIVED)
    {
        // handle indication or confirmation
        ...
    }
    else
    {
        if (rc == NO_CON_IND_RECEIVED)
        {
            // nothing received
            ...
        }
        else
        {
            //Error handling
            ...
        }
    }
    ...
}
```

### 4.4.3 Data Interface

In addition to the send/receive interface, the PROFIBUS Application Layer Interface offers a data interface which consists of data structures shared by host and controller. This data interface allows fast cyclic data transfer.

The data interface is performed by functions, which provide the data flow from and to the DPRAM area.

#### 4.4.3.1 Profi-Set-Data

Using the **profi\_set\_data** function, shared data located in the DPRAM area can be written or modified.

```
INT16 profi_set_data
(
    IN USIGN8      DataId,
    IN USIGN16     Offset,
    IN USIGN16     DataSize,
    IN VOID*       pData,
);
```

#### Function parameter description:

DataId: Identifier of the specified data structure in the Data Interface  
 Offset: Offset within the data structure  
 DataSize: Number of bytes to be written to the DPRAM  
 pData: Data block to be written

#### Possible values of data\_id (defined in the header file PB\_IF.H):

ID\_DP\_SLAVE\_IO\_IMAGE                      0x80    Identifier of image for slave I/O data (DP)

The structures of the data blocks are described in the service specific parts of the PROFIBUS User Manual.

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_SERVICE_NOT_SUPPORTED	(24)	Identifier is not supported
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_INVALID_DP_STATE	(242)	PROFIBUS interface is not in OPERATE state
- E_IF_OS_ERROR	(255)	OS system error

#### NOTES:

Writing data to the ID\_DP\_STATUS\_IMAGE is not supported in this version.

Extended error information can be obtained with `GetLastError` if the function returns `E_IF_OS_ERROR`.

## Example

```
...
#include "pb_if.h"
...
{
    USIGN16 offset;
    INT16    rc;

    // Initialize PROFIBUS API
    ...

    // Prepare and write DP data
    ...
    if (E_OK != (rc = profi_set_data(ID_DP_SLAVE_IO_IMAGE, offset, sizeof(data), &data)))
    {
        //Error handling
        ...
    }
}
```

### 4.3.3.2 Profi-Get-Data

The **profi\_get\_data** function is used to read shared data located in the DPRAM area.

```
INT16 profi_get_data
(
    IN    USIGN8      DataId,
    IN    USIGN16     Offset,
    INOUT USIGN16*    pDataSize,
    OUT   VOID*       pData
);
```

#### Function parameter description:

DataId: Identifier of the specified data structure in the Data Interface  
 Offset: Offset within the data structure  
 pDataSize: On function invocation: maximal size of the data buffer (pData)  
 On function return: number of bytes actually read  
 pData: Pointer to data buffer

#### Possible values of data\_id (defined in the header file PB\_IF.H):

ID_DP_SLAVE_IO_IMAGE	0x80	Identifier of image for slave I/O data (DP)
ID_DP_STATUS_IMAGE	0x81	Identifier of image for status data (DP)
ID_EXCEPTION_IMAGE	0xF0	Identifier of image for exception data (IF)
ID_FW_VERS_IMAGE	0xF1	Identifier of image for firmware version (IF)
ID_SERIAL_DEVICE_NUMBER	0xF2	Identifier for image for serial device number (IF)

The structures of the data blocks are described in the service specific parts (IF, DP) of the manual.

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_SERVICE_NOT_SUPPORTED	(24)	Identifier is not supported
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_INVALID_DP_STATE	(242)	PROFIBUS interface is not in OPERATE state
- E_IF_OS_ERROR	(255)	OS system error

#### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

## Example

```
...
#include "pb_if.h"
...
{
    USIGN16 offset;
    USIGN16 dataSize;
    INT16 rc;

    // Initialize PROFIBUS API
    ...
    // Read DP data
    dataSize = sizeof(data);
    if (E_OK == (rc = profi_get_data(ID_DP_SLAVE_IO_IMAGE, offset, &dataSize, &data)))
    {
        // Got data from DP slave
        ...
    }
    else
    {
        //Error handling
        ...
    }
    ...
}
```

#### 4.3.3.4 Profi-Set-Dps-Input-Data

The **profi\_set\_dps\_input\_data** function writes the input data of the DP slave to the DP-Slave input data device. It always writes the full length of the data.

```
INT16 profi_set_dps_input_data
(
    IN USIGN8*      pData,
    IN USIGN8      DataLength,
    OUT USIGN8*     pState
);
```

##### Function parameter description:

**pData:** Pointer to a USIGN8 variable containing the input data

**DataLength:** Number of input data to be written (in bytes). If the number does not correspond with the configured length of the input data, the error message 'E\_IF\_INVALID\_DATA\_SIZE' is returned.

**pState:** Pointer to the current input data status with:

- DPS\_INPUT\_STATE\_FREEZE\_ENABLED  
The slave has enabled the function for freezing the inputs.
- DPS\_INPUT\_STATE\_FREEZE\_COMMAND  
A corresponding Global\_Control command was received. Since the last time the function profi\_set\_dps\_input\_data was called the input data have been taken over as the data to be transmitted from the slave to the master. A corresponding Global\_Control command for picking up the input data was received from the master. After the execution of this function the bit is reset automatically.

##### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Too much user data
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

##### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

**Example**

```
...
#include "pb_if.h"
...
{

    USIGN8 DpsInputDataLength;           // Length of DP-Slave input data
    USIGN8 DpsInputDataState;           // DP-Slave input data state
    INT16 rc;                           // Return code

    // Initialize PROFIBUS API
    ...
    // Write input data and read recent input data state
    DpsInputDataLength = sizeof(DpsInputData);
    if (E_OK == (rc = profi_set_dps_input_data(&DpsInputData,
                                              DpsInputDataLength,
                                              &DpsInputDataState)))
    {
        // Got recent input data state
        ...
    }
    else
    {
        //Error handling
        ...
    }
    ...
}
```



#### 4.4.3.5 Profi-Get-Dps-Input-Data

The **profi\_get\_dps\_input\_data** function reads the currently set inputs and the associated status of the DP slave from the DP-Slave input data device.

```
INT16 profi_get_dps_input_data
(
    OUT   USIGN8*      pData,
    INOUT USIGN8*      pDataLength,
    OUT   USIGN8*      pState
);
```

##### Function parameter description:

**pData:** Pointer to a USIGN8 variable array to read the inputs of the slave.

**pDataLength:** (IN) Pointer to a USIGN8 variable indicating the buffer size in bytes  
(OUT) Number of input data read

**pState:** Pointer to the current input data status with:

- DPS\_INPUT\_STATE\_FREEZE\_ENABLED  
The slave has enabled the function for freezing the inputs.
- DPS\_INPUT\_STATE\_FREEZE\_COMMAND  
Since the last '**profi\_set\_dps\_input\_data**' a corresponding Global\_Control command has been received. The status is read-only. The bit will only be reset with the function **profi\_set\_dps\_input\_data**.

##### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	User buffer too small
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

##### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

**Example**

```
...
#include "pb_if.h"
...
{

    USIGN8 DpsInputDataBufferLength;           // Length of DP-Slave input data buffer
    USIGN8 DpsInputDataState;                  // DP-Slave input data state
    INT16  rc;                                 // Return code

    // Initialize PROFIBUS API
    ...
    // Read input data and recent input data state
    DpsInputDataBufferLength = sizeof(DpsInputDataBuffer);
    if (E_OK == (rc = profi_get_dps_input_data(&DpsInputDataBuffer,
                                              &DpsInputDataBufferLength,
                                              &DpsInputDataState)))
    {
        // Got input data and recent input data state
        ...
    }
    else
    {
        //Error handling
        ...
    }
    ...
}
```

#### 4.4.3.6 Profi-Get-Dps-Output-Data

The **profi\_get\_dps\_output\_data** function reads the current outputs of the DP slave from the DP-Slave output data device.

```
INT16 profi_get_dps_output_data
(
    OUT  USIGN8*    pData,
    INOUT USIGN8*    pDataLength,
    OUT  USIGN8*    pState
);
```

##### Function parameter description:

**pData:** Pointer to a USIGN8 variable array to read the outputs of the slave.

**pDataLength:** (IN) Pointer to a USIGN8 variable indicating the buffer size in bytes  
(OUT) Number of output data read

**pState:** Pointer to the current output data status with:

- **DPS\_OUTPUT\_STATE\_SYNC\_ENABLED**  
The function for freezing the outputs has been enabled.
- **DPS\_OUTPUT\_STATE\_SYNC\_COMMAND**  
A corresponding Global\_Control command was received. Since the last time the function **profi\_get\_dps\_output\_data** was called, a Sync command has been received upon which received upon which new output data have been made ready. The bit is cleared automatically after access.
- **DPS\_OUTPUT\_STATE\_CLEAR\_DATA**  
The outputs are in failsafe state. A corresponding command was received from the master.
- **DPS\_OUTPUT\_STATE\_VALID\_DATA**  
No transmission errors have occurred during data transmission from the master and user data are exchanged (no timeout or watchdog error).
- **DPS\_OUTPUT\_STATE\_NEW\_DATA**  
New output data were received from the master. Since the last access via **profi\_get\_dps\_output\_data** function new data have been delivered (independent of the Sync command). With this bit you can prevent reusing old data. The bit is cleared after access.
- **DPS\_OUTPUT\_STATE\_GLOBAL\_CONTROL**  
Since the last time the output data were read, a Global\_Control command has been received. The bit is cleared as soon as the output data have been read.

##### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	User buffer too small
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

## NOTES:

Extended error information can be obtained with GetLastError if the function returns E\_IF\_OS\_ERROR.

## Example

```
...
#include "pb_if.h"
...
{

    USIGN8 DpsOutputDataBufferLength;           // Length of DP-Slave output data buffer
    USIGN8 DpsOutputDataState;                  // DP-Slave output data state
    INT16  rc;                                  // Return code

    // Initialize PROFIBUS API
    ...
    // Read output data and current output data state
    DpsOutputDataBufferLength = sizeof(DpsOutputDataBuffer);
    if (E_OK == (rc = profi_get_dps_output_data(&DpsOutputDataBuffer,
                                                &DpsOutputDataBufferLength,
                                                &DpsOutputDataState)))
    {
        // Got output data and current output data state
        ...
    }
    else
    {
        //Error handling
        ...
    }
    ...
}
```

#### 4.4.4 Additional Interface Functions

##### 4.4.4.1 Profi-Get-Versions

The **profi\_get\_versions** function reads the version string of the PAPI dynamic link library and of the firmware on the PROFIBUS hardware.

```
INT16 profi_get_versions
(
    OUT char* pPapiVersion,
    OUT char* pFirmwareVersion,
);
```

##### Function parameter description:

pPapiVersion:            Pointer to a buffer for the version string of the PAPI DLL  
pFirmwareVersion:       Pointer to a buffer for the version string of the firmware on the PROFIBUS hardware

##### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_NO_CNTRL_RES	(10)	Cannot open board device
- E_IF_OS_ERROR	(255)	OS system error

##### NOTES:

Both buffers for the version strings must have at least the size of **VERSION\_STRING\_LENGTH**. The PROFIBUS API does not have to be initialized to get to get the version strings. Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

##### Example

```
...
#include "pb_if.h"
...
{
    char  papiVersion[VERSION_STRING_LENGTH];
    char  firmwareVersion[VERSION_STRING_LENGTH];
    INT16 rc;

    if (E_OK == (rc = profi_get_versions(papiVersion,firmwareVersion)))
    {
        // Got the version strings
    }
}
```

## 4.4.4.2 Profi-Get-Serial-Device-Number

The **profi\_get\_serial\_device\_number** function reads the serial device number of the PROFIBUS hardware.

```
INT16 profi_get_serial_device_number
(
    OUT USIGN32 * pSerialDeviceNumber
);
```

### Function parameter description:

pSerialDeviceNumber:            Pointer to the variable for serial device number

**Possible function return values** (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_NO_CNTRL_RES	(10)	Cannot open board device
- E_IF_OS_ERROR	(255)	OS system error

### NOTES:

**The PROFIBUS API does not have to be initialized to get to get the serial device number. Extended error information can be obtained with GetLastError if the function returns E\_IF\_OS\_ERROR.**

### Example

```
...
#include "pb_if.h"
...
{
    USIGN32 serialDeviceNumber;
    INT16 rc;

    if (E_OK == (rc = profi_get_serial_device_number(&serialDeviceNumber)))
    {
        // Got the serial device number
    }
}
```

## 4.5 ENHANCED PROFIBUS APPLICATION PROGRAM INTERFACE

The enhanced PROFIBUS Application Program Interface contains a set of new functions. These functions have similar functionality as the standard PROFIBUS Application Program Interface functions, but they use Win32 device handles as parameters for describing where to read or write frames or data.

This interface encapsulates the whole functionality of the protocol kernel device driver. You can use all devices provided by this driver with the functions of the interface.

The enhanced PROFIBUS API consists of these functions:

<b>profi_open_basic_management</b>	Open basic management device
<b>profi_open</b>	Open device
<b>profi_close</b>	Close device
<b>profi_write_service</b>	Send frame
<b>profi_read_service</b>	Receive frame
<b>profi_read_multi</b>	Receive frame from device array
<b>profi_write_data</b>	Write DP slave data
<b>profi_read_data</b>	Read DP slave data
<b>profi_get_cntrl_info</b>	Read version string of firmware Read serial device number
<b>profi_set_timeout</b>	Set time-out
<b>profi_get_timeout</b>	Get time-out
<b>profi_set_queue_size</b>	Set queue size
<b>profi_get_queue_size</b>	Get queue size
<b>profi_get_overrun_count</b>	Get overrun count

## 4.5.1 Profi-Open-Basic-Management

The **profi\_open\_basic\_management** function is used to initialize the enhanced PROFIBUS API and opens the basic management device of the desired board.

```
HANDLE profi_open_basic_management
(
    IN USIGN8    Board,
    IN USIGN8    Channel,
    IN INT32     DesiredAccess
);
```

### Function parameter description:

Board	Number (between 0 and 9) of the board to work on.						
Channel	PROFIBUS channel number. This version supports only channel 0						
DesiredAccess	Specifies the type of access to the basic management device. An application can obtain read access, write access or read-write access. You can use the following flag constants to build a value for this parameter. Both GENERIC_READ and GENERIC_WRITE must be set to obtain read/write access. If DesiredAccess is 0, neither read nor write access is allowed; only IOControl operations that do not need a specific access right can be performed on the device.						
<table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>GENERIC_READ</td><td>Specifies read access to the device. Data can be read from the device.</td></tr> <tr> <td>GENERIC_WRITE</td><td>Specifies write access to the device. Data can be written to the device.</td></tr> </table>		Value	Meaning	GENERIC_READ	Specifies read access to the device. Data can be read from the device.	GENERIC_WRITE	Specifies write access to the device. Data can be written to the device.
Value	Meaning						
GENERIC_READ	Specifies read access to the device. Data can be read from the device.						
GENERIC_WRITE	Specifies write access to the device. Data can be written to the device.						

### Possible function return values:

- If the functions succeeds, the return value is the open handle of the basic management device.
- If the functions fails, the return value is INVALID\_HANDLE\_VALUE. To obtain extended error information, call **GetLastError**.

### NOTES:

If more than one board is used, **profi\_open\_basic\_management** must be called for each board to obtain the handle of the basic management device.



### Example

```
...
#include "pb_if.h"
...
{
    HANDLE hBasicMgmt;
    USIGN8 boardNr;

    ...

    if (INVALID_HANDLE_VALUE == (hBasicMgmt = profi_open_basic_management(boardNr, 0,
                                                                           GENERIC_READ)))
    {
        ... // error handling
    }
}
```

### 4.5.2 Profi-Open

Use the **profi\_open** function to open any device of the protocol device driver besides the basic management device. This function checks if it need to create the device with an IoControl call to the basic management device and than opens the device with the desired access rights.

HANDLE profi\_open

```
(
    IN HANDLE    hManagement,
    IN INT32     DeviceType,
    IN USIGN32    Index,
    IN INT32     DesiredAccess
);
```

#### Function parameter description:

**hManagement:** Handle to the management device of the desired board. The basic management device can be used to open all device types. The DP management device is only able to open DP devices. The FDL management device opens only FDL SAP devices and the FMS management device opens only FMS CR devices.

**DeviceType:** Type of the device to open. You can open any device of the protocol device driver.

Value	Meaning
DEVICE_DP_MANAGEMENT	DP management device
DEVICE_DP_SERVICE	DP service device
DEVICE_DP_SLAVE_DATA	DP slave data device
DEVICE_DP_MSAC	DP master slave acyclic device
DEVICE_FDL_MANAGEMENT	FDL management device
DEVICE_FDL_SAP	FDL SAP device
DEVICE_FMS_MANAGEMENT	FMS management device
DEVICE_FMS_CR	FMS CR device

**Index:** If more than one device of a device type can be opened, the index (zero based) of the device must be passed. You can open: 128 DP service, DP master slave acyclic and DP slave data devices, 64 FDL SAP devices and 48 FMS CR devices. This parameter is ignored for all management devices.

**DesiredAccess** Specifies the type of access to the device. An application can obtain read access, write access or read-write access. The following flag constants can be used to build a value for this parameter. Both GENERIC\_READ and GENERIC\_WRITE must be set to obtain read-write access. If DesiredAccess is 0, neither read nor write access is allowed; only IOControl operations that do not need a specific access right can be performed on the device.

Value	Meaning
GENERIC_READ	Specifies read access to the device
GENERIC_WRITE	Specifies write access to the device.

### Possible function return values:

- If the function succeeds, the return value is the open handle of the device.
- If the functions fails, the return value is INVALID\_HANDLE\_VALUE. To obtain extended error information, call **GetLastError**.

### Example

```
...
#include "pb_if.h"
#include "devtypes.h"
...
{
    HANDLE hBasicMgmt;
    HANDLE hCR34;

    // Open the basic management device
    ...

    // Open the FMS CR 34 device for read and write access
    if (INVALID_HANDLE_VALUE == (hCR34 = profi_open(hBasicMgmt, DEVICE_FMS_CR, 34,
                                                    GENERIC_READ | GENERIC_WRITE)))
    {
        // error handling
        ...
    }
    ...
}
```

### 4.5.3 Profi-Close

The **profi\_close** function is used to close any device opened with **profi\_open\_basic\_management** or **profi\_open**.

```
BOOL profi_close  
(  
    IN HANDLE hDevice  
);
```

#### Function parameter description:

hDevice:                      Handle of the device to close

#### Possible function return values:

- If the functions succeeds, the return value is **TRUE**.
- If the functions fails, the return value is **FALSE**. To obtain extended error information, call **GetLastError**.

#### Example

```
...  
#include "pb_if.h"  
...  
{  
    HANDLE handle;  
  
    // open the device  
    ...  
    if (!profi_close(handle))  
    {  
        // error handling  
        ...  
    }  
}
```

#### 4.5.4 Profi-Write-Service

The **profi\_write\_service** function writes a service frame to a service-oriented device. The frame could be a service request or a response to a received indication.

```
INT16 profi_write_service
(
    IN HANDLE                hDevice,
    IN T_PROFI_SERVICE_DESCR* pSdb,
    IN VOID*                 pData
);
```

##### Function parameter description:

hDevice	Handle of the service-oriented service device
pSdb	Pointer to the description of the service to send
pData	Pointer to the service-specific parameters and data

##### Possible function return values(defined in the header file PB\_ERR.H):

- E_OK	(0)	Function executed correctly
- E_IF_FATAL_ERROR	(7)	Unrecoverable error on PROFIBUS controller
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_INVALID_LAYER	(12)	Invalid layer
- E_IF_INVALID_SERVICE	(13)	Invalid service identifier
- E_IF_INVALID_PRIMITIVE	(14)	Invalid service primitive
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_INVALID_COMMREF	(16)	Invalid communication reference
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_RESOURCE_UNAVAILABLE	(21)	No resource available
- E_IF_NO_PARALLEL_SERVICES	(22)	No parallel services allowed
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service temporarily not executable
- E_IF_SERVICE_NOT_SUPPORTED	(24)	Service not supported in subset
- E_IF_SERVICE_NOT_EXECUTABLE	(25)	Service not executable
- E_IF_INVALID_PARAMETER	(30)	Invalid parameter in REQ or RES
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

##### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

If **profi\_write\_service** fails with **E\_IF\_NO\_CTRL\_RES** the controller hasn't responded during the write time-out time of the service devices. You can set the time-out value by calling **profi\_set\_timeout**.

You can only write services to a device which are supported by the device. See the device description to check if the device supports the service. You can't write any services to the data oriented DP slave data devices. The service device pointed to by **hDevice** must be opened with the function **profi\_open** and write access rights

## Example

```
#include "pb_if.h"
...
{
    HANDLE                hCR3;                // Handle to FMS CR 3 device
    T_PROFI_SERVICE_DESCR sdb;                // Service description
    T_VAR_READ_REQ        readReq;            // Read request
    INT16                 rc;
    USIGN8                invokeId;

    // Open the handle to FMS CR 3 device
    ...

    // Send a FMS_READ request

    // Fill the service description
    sdb.layer              = FMS;
    sdb.service            = FMS_READ;
    sdb.primitive          = REQ;
    sdb.invoke_id          = ++invokeId;
    sdb.comm_ref           = 5; // Send it to CR 5

    // fill the read request
    readReq.acc_spec.tag    = ACCESS_INDEX;
    readReq.acc_spec.id.index = 20;
    readReq.subindex        = 0;

    if (rc = profi_write_service(hCR3, &sdb, (void *)&readReq))
    {
        // Error handling
        ...
    }
    ...
}
```

### 4.5.5 Profi-Read-Service

The **profi\_read\_service** function reads the oldest received frame from a service-oriented device. The frame could be a service indication or a confirmation for a requested service.

INT16 profi\_read\_service

```
(
    IN    HANDLE                hDevice,
    OUT   T_PROFI_SERVICE_DESCR* pSdb,
    OUT   VOID*                 pData,
    INOUT USIGN16*               pDataLen,
);
```

**Function parameter description:**

hDevice	Handle of the service-oriented device
pSdb	Pointer to the buffer for the received service description
pData	Pointer to the buffer for the received service-specific parameters and data
pDataLen	On function invocation: size, in bytes, of the buffer pointer by data. On function return: size of the received service-specific data.

The function returns **CON\_IND\_RECEIVED** to signal that a confirmation or indication is available.

**Possible function return values** (defined in the header file PB\_ERR.H):

- NO_CON_IND_RECEIVED	(0)	There is no confirmation or indication
- CON_IND_RECEIVED	(1)	Confirmation or indication is available
- E_IF_FATAL_ERROR	(7)	Unrecoverable error on PROFIBUS controller
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_INVALID_DATA_SIZE	(15)	Size of data block provided not sufficient
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_RESOURCE_UNAVAILABLE	(21)	No resource available
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

### NOTES:

Extended error information can be obtained with `GetLastError` if the function returns **E\_IF\_OS\_ERROR**.

`profi_read_service` returns after a frame was received or the read service time-out elapsed. The time-out value can be set by calling `profi_set_timeout`. The service device pointed to by `hDevice` must be opened with the function `profi_open` and read access rights.

## Example

```
...
#include "pb_if.h"
...
{
    HANDLE                hCR3;                // Handle to FMS CR 3 device
    T_PROFI_SERVICE_DESCR sdb;                // Service
description
    BYTE                data[MAX_FMS_PDU_LENGTH];                // Data
buffer
    USIGN16                dataLen;
    INT16                rc;

    // Open the handle to FMS CR 3 device
    ...

    // Read a received frame from FMS CR 3
    dataLen = sizeof(data);
    rc = profi_read_service(hService, &sdb, &data, &dataLen);

    switch(rc)
    {
        case CON_IND_RECEIVED:
        {
            // frame received
            ...
        }
        case NO_CON_IND_RECEIVED:
        {
            // No frame received
            ...
        }

        default:
        {
            //Error handling
            ...
        }
    }
    ...
}
```



#### 4.5.6 Profi-Read-Multi

The **profi\_read\_multi** function reads the oldest received service frame from an array of service-oriented devices. The frame could be a service indication or a confirmation for a requested service.

```
INT16 profi_read_multi
(
    OUT  T_PROFI_SERVICE_DESCR* pSdb,
    OUT  VOID*                   pData,
    INOUT USIGN16*               pDataLen,
    IN    USIGN16                NrOfHandles,
    IN    HANDLE*                phDevices,
);
```

##### Function parameter description:

pSdb	Pointer to the buffer for the service description block
pData	Pointer to the buffer to receive service-specific parameters and data
pDataLen	On function invocation: size, in bytes, of the buffer pointer by data. On function return: size of the received service-specific data.
NrOfHandles	Number of device handles in the device array pointed by pHandlesService.
phDevices	Pointer to an array of service device handles.

The function returns **CON\_IND\_RECEIVED** to signal that a confirmation or indication is available.

##### Possible function return values (defined in the header file PB\_ERR.H):

- NO_CON_IND_RECEIVED	(0)	There is no confirmation or indication
- CON_IND_RECEIVED	(1)	Confirmation or indication is available
- E_IF_FATAL_ERROR	(7)	Unrecoverable error on PROFIBUS controller
- E_IF_NO_CNTRL_RES	(10)	Controller does not respond (CMI_TIMEOUT)
- E_IF_INVALID_DATA_SIZE	(15)	Size of data block provided not sufficient
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_RESOURCE_UNAVAILABLE	(21)	No resource available
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_OS_ERROR	(255)	OS system error

## NOTES:

Extended error information can be obtained with `GetLastError` if the function returns `E_IF_OS_ERROR`.

The function checks every device in the array, beginning with the first if a frame has been received. If a device received a frame, `profi_read_multi` returns with the received frame. If more than one device received a frame, the oldest received frame is returned.

`profi_read_multi` returns after a frame was received or the read service time-out elapsed. The time-out value can be set by calling `profi_set_timeout`.

All service devices in the array pointed to by *phDevices* must be opened with the function `profi_open` and read access rights

## Example

```
...
#include "pb_if.h"
...
{
    HANDLE                hDevs[6];                // Device handle array
    T_PROFI_SERVICE_DESCR sdb;                    // Service description
    BYTE                  data[MAX_FMS_PDU_LENGTH]; // Data buffer
    USIGN16               dataLen;
    INT16                 rc;
    int                   i;

    // open basic management of board 0
    if (INVALID_HANDLE_VALUE == (hDevs[0]=profi_open_basic_management(0,0,GENERIC_READ)))
    {
        // Error handling
        ...
    }

    // open the handles to FMS CR2 - CR6 devices
    for (i = 2; i <= 6; i++)
    {
        if (INVALID_HANDLE_VALUE == (hDevs[i-1] = profi_open(hDevs[0], DEVICE_FMS_CR, i,
                                                                GENERIC_READ | GENERIC_WRITE)))
        {
            // Error handling
            ...
        }
    }
    ...

    // Read a received frame from the devices in the device array
    dataLen = sizeof(data);
}
```

```
rc = profi_read_multi(&sdb, &data, &dataLen, 6, hDevs);

switch(rc)
{
    case CON_IND_RECEIVED:
    {
        // frame received
        ...
    }

    case NO_CON_IND_RECEIVED:
    {
        // No frame received
        ...
    }

    default:
    {
        //Error handling
        ...
    }
}
}
```

### 4.5.7 Profi-Write-Data

The **profi\_write\_data** function writes data to a DP slave data device which is used to access the I/O area of the DP slave.

```
INT16 profi_write_data
(
    IN HANDLE          hDevice,
    IN VOID*           pData,
    IN USIGN16         DataLen
);
```

#### Function parameter description:

hDevice	Handle of the DP slave data device to write the data
pData	Data block to be written
DataLen	Size, in bytes, of the data block pointed to by data

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service not executable at time
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_SLAVE_ERROR	(240)	No valid communication with the slave
- E_IF_SLAVE_DIAG_DATA	(241)	New diagnostic data available
- E_IF_INVALID_DP_STATE	(242)	PROFIBUS interface is not in OPERATE state
- E_IF_OS_ERROR	(255)	OS system error

#### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

### Example

```
...
#include "pb_if.h"
...
{
    HANDLE hData3;
    INT16 rc;

    // Open the DP slave data device of the slave 3
    ...

    // Write DP data
    if (rc = profi_write_data(hData3, &data, sizeof(data)))
    {
        //error handling
        ...
    }
    ...
}
```

### 4.5.8 Profi-Read-Data

The **profi\_read\_data** function reads data from a DP slave data device which is used to access the I/O area of the DP slave.

```
INT16 profi_read_data
(
    IN    HANDLE    hDevice,
    OUT   VOID*      pData,
    INOUT USIGN16*   pDataLen
);
```

#### Function parameter description:

hDevice	Handle of the DP slave data device to read the data
pData	Pointer to the buffer to store the read data
pDataLen	On function invocation: size, in bytes, of the data block pointed to by data On function return: number of bytes read into data

#### Possible function return values (defined in the header file PB\_ERR.H):

- E_OK	(00)	Function executed correctly
- E_IF_INVALID_DATA_SIZE	(15)	Not enough CMI data block memory
- E_IF_CMI_ERROR	(20)	Serious CMI error
- E_IF_SERVICE_CONSTR_CONFLICT	(23)	Service not executable at time
- E_IF_PAPI_NOT_INITIALIZED	(33)	API not initialized
- E_IF_SLAVE_ERROR	(240)	No valid communication with the slave
- E_IF_SLAVE_DIAG_DATA	(241)	New diagnostic data available
- E_IF_INVALID_DP_STATE	(242)	PROFIBUS interface is not in OPERATE state
- E_IF_OS_ERROR	(255)	OS system error

#### NOTES:

Extended error information can be obtained with **GetLastError** if the function returns **E\_IF\_OS\_ERROR**.

## Example

```
...
#include "pb_if.h"
...
{
    USIGN16 dataSize;
    HANDLE  hData3;
    INT16   rc;

    // Open the DP slave data device of the slave 3
    ...

    // read DP data
    dataSize = sizeof(data);
    if ((rc = profi_read_data(hData3,&data,&dataSize)) == E_OK)
    {
        // Received data from DP slave 3
        ...
    }
    else
    {
        //Error handling
        ...
    }
    ...
}
```

### 4.5.9 Profi-Get-Cntrl-Info

The **profi\_get\_cntrl\_info** function reads the version of the firmware and the serial device number of PROFIBUS interface.

```

BOOL profi_get_cntrl_info
(
    IN    USIGN8          BoardNr,
    OUT   char*            pFwVersion
    OUT   USIGN32*         pSerialDeviceNumber
);

```

#### Function parameter description:

BoardNr	Number of the board to read the version information
pFwVersion	Pointer to the buffer for the version string of the firmware on the board
pSerialDeviceNumber	Pointer to the serial device number

**Possible function return values** (defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. Extended error information can be obtained with GetLastError.

#### NOTES:

**The buffer for the version string must have at least the size of VERSION\_STRING\_LENGTH.**

#### Example

```

...
#include "pb_if.h"
...
{
    USGIN8    boardNr;
    UNSIGN32  serialDeviceNumber;
    char      firmwareVersion[VERSION_STRING_LENGTH];

    ...
    if (profi_get_fw_version(boardNr, firmwareVersion, &serialDeviceNumber))
    {
        // Received the firmware version string
        ...
    }
}

```



#### 4.5.10 Profi-Set-Timeout

The **profi\_set\_timeout** function is used to set the time-out values for service-oriented read and write operations performed by the functions **profi\_write\_service**, **profi\_read\_service** and **profi\_read\_multi**.

```
BOOL profi_set_timeout
(
    IN HANDLE          hBasicMgmtDevice,
    IN USIGN32         ReadTimeout,
    IN USIGN32         WriteTimeout
);
```

##### Function parameter description:

hBasicMgmtDevice	Handle of the basic management device
ReadTimeout	Maximum duration of a read operation in milliseconds (WAIT_FOREVER for infinite wait).
WriteTimeout	Maximum duration of a write operation in milliseconds (WAIT_FOREVER for infinite wait).

##### Possible function return values (defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. Extended error information can be obtained with **GetLastError**.

##### NOTES:

The time-out set on one service device is used for all service devices of this board. Two different time-outs cannot be set on one board.

A polling mode (do not wait) can be implemented by setting the time-out values to 0.

The default time-out values are 0.

The service device pointed to by *hBasicMgmtDevice* must be opened with read access rights.

## Example

```
...
#include "pb_if.h"
...
{
    HANDLE hBasicMgmt;

    // Open the basic management device
    ...
    // Set time-out of 30 milliseconds
    if (!profi_set_timeout(hBasicMgmt,30,30))
    {
        // set timeouts failt
    }
    ...
}
```

#### 4.5.11 Profi-Get-Timeout

The **profi\_get\_timeout** function is used to get the time-out values for service-oriented read and write operations performed by the functions `profi_write_service` , `profi_read_service` and `profi_read_multi`.

```
BOOL profi_get_timeout
(
    IN    HANDLE      hBasicMgmtDevice,
    OUT   USIGN32*     pReadTimeout,
    OUT   USIGN32*     pWriteTimeout
);
```

##### Function parameter description:

<code>hBasicMgmtDevice</code>	Handle of the basic management device
<code>pReadTimeout</code>	Maximum duration of a read operation in milliseconds.
<code>pWriteTimeout</code>	Maximum duration of a write operation in milliseconds.

##### Possible function return values (defined in the header file `PB_ERR.H`):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. Extended error information can be obtained with **GetLastError**.

#### Example

```
#include "pb_if.h"
...
{
    HANDLE  hBasicMgmt;
    USIGN32 readTimeout, writeTimeout;

    // Open the basic management device
    ...

    // get timeout values
    if (profi_get_timeout(hBasicMgmt,&readTimeout,&writeTimeout))
    {
        // Received time-out values
    }
    ...
}
```

#### 4.5.12 Profi-Set-Queue-Size

The **profi\_set\_queue\_size** function is used to set the maximum size of the queue for received frames of the protocol driver. If the size of the queue exceeds the maximum size, a received frame will be ignored. The number of ignored frames is counted in an overrun count which can be obtained by calling **profi\_get\_overrun\_count**

```

BOOL profi_set_queue_size
(
    IN HANDLE    hBasicMgmtDevice,
    IN USIGN32   QueueSize
);

```

##### Function parameter description:

hBasicMgmtDevice	Handle of the basic management device
QueueSize	Maximum size of the frame queue for received frames.

**Possible function return values** (defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. Extended error information can be obtained with **GetLastError**.

##### NOTES:

The received frames queue is used to store received frames in the driver, which are not read by calling **profi\_read\_service** or **profi\_read\_multi**.

The maximum size of the received frames queue can be set for each board. The default maximum size of the queue is 32.

##### Example

```

#include "pb_if.h"

{
    HANDLE hBasicMgmt;

    // Open the basic management device
    ...

    // Set queue size to 100
    if (!profi_set_queue_size(hBasicMgmt,100))
    {
        // set queue size failt
    }
    ...
}

```

#### 4.5.13 Profi-Get-Queue-Size

The **profi\_get\_queue\_size** function is used to obtain the maximum size of the queue for received frames of the protocol driver.

```
BOOL profi_get_queue_size
(
    IN    HANDLE    hBasicMgmtDevice,
    OUT   USIGN32*   pQueueSize
);
```

##### Function parameter description:

hBasicMgmtDevice	Handle of the basic management device
pQueueSize	Maximum size of the frame queue for received frames

**Possible function return values** (defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. Extended error information can be obtained with **GetLastError**.

##### NOTES:

The received frames queue is used to store received frames in the driver, which are not read by calling **profi\_read\_service** or **profi\_read\_multi**.

##### Example

```
#include "pb_if.h"

{
    HANDLE hBasicMgmt;
    USIGN32 queueSize;
    // Open the basic management device
    ...
    // Set queue size to 100
    if (!profi_get_queue_size(hBasicMgmt, &queueSize))
    {
        // get queue size fault
    }
    ...
}
```

#### 4.5.14 Profi-Get-Overrun-Count

The **profi\_get\_overrun\_count** function is used to get the overrun count of the received frames queue of the protocol driver. This value counts the number of frames arrived while the size of the received frames queue exceeded the maximum queue size. Each call to this function resets the overrun count to 0.

```

BOOL profi_get_overrun_count
(
    IN HANDLE    hBasicMgmtDevice,
    IN USIGN32*  pOverrunCount
);

```

##### Function parameter description:

hBasicMgmtDevice	Handle of the frame-oriented service device
pOverrunCount	Number of ignored received frames

##### Possible function return values (defined in the header file PB\_ERR.H):

- If the function succeeds, the return value is **TRUE**.
- If the function fails, the return value is **FALSE**. Extended error information can be obtained with **GetLastError**.

##### NOTES:

The received frames queue is used to store received frames in the driver, which are not read by calling **profi\_read\_service** or **profi\_read\_multi**.

##### Example

```

#include "pb_if.h"
...
{
    HANDLE  hBasicMgmt;
    USIGN32 overrunCount;

    // Open the basic management device
    ...

    // Get overrun count
    if (profi_get_overrun_count(hBasicMgmt,&overrunCount))
    {
        // Received overrun count
    }
    ...
}

```

## 4.6 INTERFACE RETURN VALUES

This chapter gives an overview of the user interface return values. All possible return values are described in the header files PB\_IF.H and PB\_ERR.H.

### Overview of User Interface error codes and return values

Identifier	Value	Description
- E_OK	0	No error occurred
- NO_CON_IND_RECEIVED	0	No confirmation or indication available
- CON_IND_RECEIVED	1	Confirmation or indication ws received
- E_IF_FATAL_ERROR	7	Unrecoverable error on board <sup>1)</sup>
- E_IF_INIT_INVALID_PARAMETER	8	Invalid initialization parameter
- E_IF_NO_CNTRL_RES	10	Controller does not respond
- E_IF_INVALID_CNTRL_TYPE_VERSION	11	Invalid controller type or invalid firmware version
- E_IF_INVALID_LAYER	12	Invalid layer
- E_IF_INVALID_SERVICE	13	Invalid service identifier
- E_IF_INVALID_PRIMITIVE	14	Invalid service primitive
- E_IF_INVALID_DATA_SIZE	15	Not enough CMI data block memory
- E_IF_INVALID_CMI_CALL	19	Invalid CMI call
- E_IF_CMI_ERROR	20	Error occurred in CMI
- E_IF_RESOURCE_UNAVAILABLE	21	No resource available
- E_IF_NO_PARALLEL_SERVICES	22	No parallel services allowed
- E_IF_SERVICE_CONSTR_CONFLICT	23	Service temporarily not executable
- E_IF_SERVICE_NOT_SUPPORTED	24	Service not supported
- E_IF_SERVICE_NOT_EXECUTABLE	25	Service not executable
- E_IF_INVALID_ACCESS	26	Invalid access to protocol software
- E_IF_NO_CNTRL_PRESENT	28	No controller present
- E_IF_INVALID_PARAMETER	30	Invalid parameter in REQ or RES
- E_IF_INIT_FAILED	31	Init. API or Controller failed
- E_IF_EXIT_FAILED	32	Exit API or Controller failed
- E_IF_PAPI_NOT_INITIALIZED	33	API not initialized
- E_IF_SLAVE_DIAG_DATA	240	no data available
- E_IF_SLAVE_ERROR	241	no data exchange
- E_IF_INVALID_DP_STATE	242	DP is not in state clear/operate
- E_IF_READING_REGISTRY	243	Error reading registry
- E_IF_OS_ERROR	255	OS system (WIN,DOS) error

- 1) **NOTE:** If the interface error **E\_IF\_FATAL\_ERROR** is indicated, the User can read additional information about this error via the service interface function **profi\_rcv\_con\_ind** or data interface function **profi\_get\_data**:

## Read additional error information via **profi\_rcv\_con\_ind**:

### *Service-Description-Block for Indication:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_EXCEPTION
USIGN8	primitive	IND
INT8	invoke_id	0
INT16	result	POS

### *Data block for Indication:*

Data structure      T\_EXCEPTION

USIGN8	task_id	Task in wich the fatal system error is occurred
USIGN8	par1	Exception parameter 1
USIGN16	par2	Exception parameter 2
USIGN16	par3	Exception parameter 3

## Read additional error information via **profi\_get\_data**:

```
profi_get_data (ID_EXCEPTION_IMAGE,      /* Identifier of the exception description */
               0,                        /* Offset in the exception description */
               (USIGN16 FAR*) &data_len, /* Size of the exception description */
               (T_EXCEPTION FAR*) &exception /* Pointer to the exception description */
               );
```

```
T_EXCEPTION      exception;               /* Defined in the header file PB_ERR.H */
USIGN16          data_descr_len = sizeof(T_EXCEPTION);
```



# **PROFIBUS Application Program Interface**

## **Basic Management**

Version 5.2  
Rev. 02

Date: 08-April-1999

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 - 45 65 6 - 0  
Fax (++49) 89 - 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---

## CONTENTS

1 SCOPE .....	1
2 OVERVIEW .....	3
3 FMB SERVICES .....	6
3.1 FMB-Set-Configuration .....	6
3.2 FMB-Set-Value Services .....	12
3.2.1 FMB-Set-Busparameter .....	13
3.2.2 FMB-Set-Value .....	15
3.3 FMB-Read-Value Services .....	16
3.3.1 FMB-Read-Busparameter .....	17
3.3.2 FMB-Read-Value .....	18
3.4 FMB-LSAP-Status .....	20
3.5 FMB-Get-Live-List .....	22
3.6 FMB-FM2-Event .....	24
3.7 FMB-Reset .....	25
3.8 FMB-Exit .....	26
3.9 FMB-Exception .....	27
4 CONFIGURATION PARAMETERS .....	28
4.1 FMB Configuration .....	28
4.1.1 VFD Configuration .....	28
4.1.2 CRL Configuration .....	29
4.1.2.1 Buffers for a Master-Master CR .....	29
4.1.2.2 Buffers for Masters in Master-Slave CRs .....	30
4.1.2.3 Buffers for Slaves in Master-Slave CRs .....	31
4.1.2.4 Buffers for connectionless CRs .....	31
4.1.3 DP Configuration .....	32
4.1.4 FDLIF Configuration .....	32
4.1.5 Standard Configuration .....	33
4.2 FDL Bus Parameters .....	34
4.2.1 Range of Values .....	34
4.2.2 Recommended Bus Parameters for FMS Operation .....	37
4.2.3 Recommended Bus Parameters for FMS Operation using ASPC2 .....	38
4.2.4 Recommended Bus Parameters for DP and FMS Operation .....	39
4.2.5 Recommended Bus Parameters for DP Operation .....	40
APPENDIX A .....	41
STANDARD ERROR STRUCTURE AND ERROR CODES .....	41

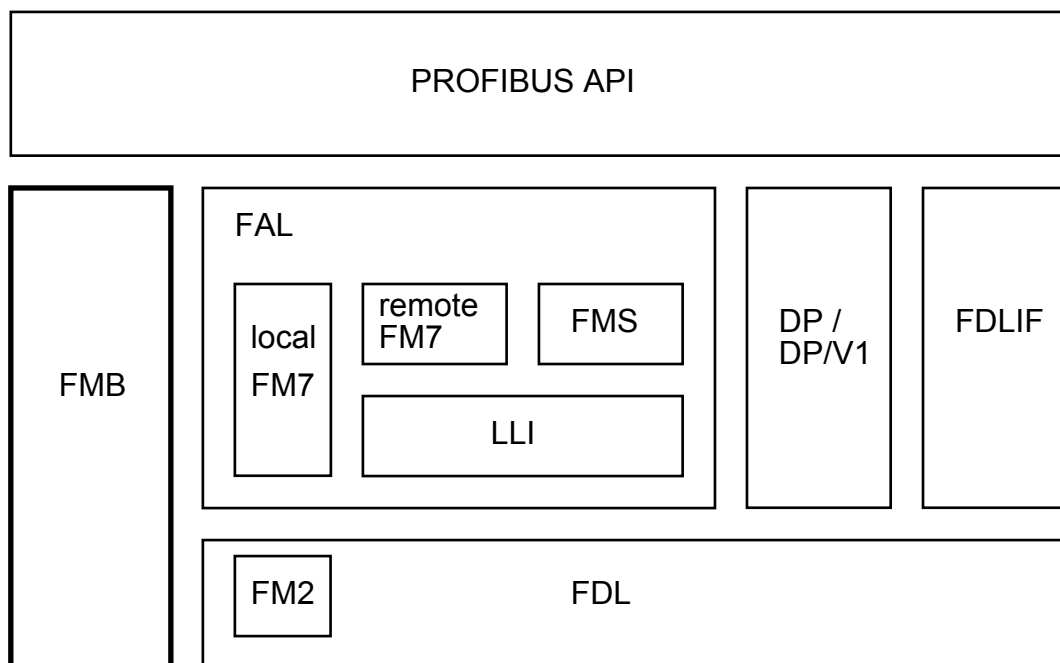


## 1 SCOPE

Softing's PROFIBUS protocol software is designed to allow mixed operation of the protocol components FAL, DP and FDLIF. These protocol components compete for the available memory and for access to FDL. In order to coordinate the mixed operation of FAL, DP and FDLIF, a new management component was introduced in the protocol stack. This management component is designated Basic Management (FMB).

Softing's PROFIBUS Application Program Interface provides uniform access to all service groups of the PROFIBUS protocol. The common access functions are described in the "User Interface" part of the PROFIBUS User Manual.

This document describes the specific services and configuration parameters of the Basic Management (FMB).



This document should be read in conjunction with the following parts of the PROFIBUS User Manual:

- "User Interface" Describes the uniform access functions to all PROFIBUS services
- "FM7 Services" Describes the management services which are necessary to configure the Fieldbus Application Layer (FAL)
- "DP Services" Describes the services which are necessary to configure the Decentral Periphery (DP)
- "DP/V1 Services" Describes the services which are necessary to configure the Decentral Periphery (DP/V1)



## 2 OVERVIEW

The main goal of FMB is to configure the PROFIBUS protocol stack and to start FDL. For this reason, FMB provides the services:

- FMB-Set-Configuration
- FMB-Set-Busparameter

Furthermore FMB provides services to set and read FDL parameters, to request LSAP configuration and read the Live-List. This services are:

- FMB-Read-Busparameter
- FMB-Set-Value, FMB-Read-Value
- FMB-LSAP-Status
- FMB-Get-Live-List

To stop of the whole PROFIBUS protocol stack FMB provides the servcies:

- FMB-Reset
- FMB-Exit

In addition FMB indicates FM2 events and system fatal errors to the PROFIBUS user. The services for indicating those events and errors are:

- FMB-FM2-Event
- FMB-Exception

**Overview of FMB services**

<b>Service group</b>	<b>Identifier</b>	<b>Code</b>	<b>Page</b>
System configuration	FMB_SET_CONFIGURATION	27	6
Set and read FDL Bus Parameters	FMB_SET_BUSPARAMETER	22	13
	FMB_READ_BUSPARAMETER	24	17
Set and read a single FDL parameter	FMB_SET_VALUE	15	15
	FMB_READ_VALUE	16	18
Read status of FDL SAP	FMB_LSAP_STATUS	17	20
Read Live-List	FMB_GET_LIVE_LIST	26	22
Event Indications from FM2	FMB_FM2_EVENT	19	24
System fatal error indication	FMB_EXCEPTION	28	27
System halt and restart	FMB_EXIT	21	26
	FMB_RESET	20	25



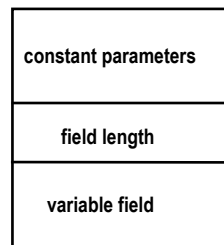
## Notes on Data Structures and Parameters

The FMB-specific types and constants are defined in the include file PB\_FMB.H.

All words, long-words, strings, arrays and records begin on even addresses. To accomplish this, fill bytes had to be added in some places. These are always recognizable by their name *dummy*.

Data blocks do not contain pointers. If a data block contains one or more fields or lists of variable length, then the length information of all variable-length fields is stored in the constant part. The fields of variable length follow on the constant part.

Here is an example of such a data block:



The variable data fields are shown between comment delimiters in the include file PB\_FMB.H to show their position and structure, without forcing the programmer to use data structures of a specific length. Nevertheless, the data must be entered at exactly this spot.

The service description block contains a *result* parameter. If a function returns as positive (result = POS) the service-specific confirmation block will be passed. If the result is negative (result = NEG), then the error structure T\_ERROR or a service-specific error structure is passed.

For negative confirmations a standard error structure T\_ERROR is used. The error codes are not noted explicitly for each service. The standard error structure and the error codes are described in Appendix A.

### 3 FMB SERVICES

#### 3.1 FMB-Set-Configuration

This service is used to configure the PROFIBUS protocol stack. In the service request the PROFIBUS user specifies which protocol components are active and what is the available memory for the active protocol components.

The FMB-Set-Configuration service must be executed directly after initialization of the protocol stack. It is not allowed to execute any other service prior to the configuration service. If the PROFIBUS user starts with any other service (e.g. FM7-Set-Busparameter), FMB assumes a standard configuration. In this standard configuration only FAL is active.

Any attempt to execute the FMB-Set-Configuration service after another service and any attempt to execute this service a second time is confirmed with a negative result.

##### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_SET_CONFIGURATION
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

##### *Data block for Request:*

Data structure	T_FMB_SET_CONFIGURATION_REQ	
PB_BOOL	fms_active	FMS- and FM7-Services usable
PB_BOOL	dp_active	DP-Services are usable
PB_BOOL	fdlif_active	FDLIF-Services are usable
PB_BOOL	sm7_active	reserved
USIGN16	fdl_evt_receiver	receiver of FM2 events is FMB_USR, FM7_USR, DP_USR or FDLIF_USR (1)
USIGN16	data_buffer_length	max. size of PDU buffer
T_FMB_CONFIG_VFD	vfd	VFD configuration parameters (FMS)
T_FMB_CONFIG_CRL	crl	CRL configuration parameters (FM7)
T_FMB_CONFIG_DP	dp	DP configuration parameters
T_FMB_CONFIG_FDLIF	fdlif	FDLIF configuration parameters
T_FMB_CONFIG_SM7	sm7	reserved

(1) In future releases of SOFTING's PROFIBUS API, only the receiver FMB\_USR will be supported. Other receivers are supported only for compatibility with former releases of PROFIBUS API. In new applications always set FMB\_USR.

### VFD-Configuration

#### Data structure T\_FMB\_CONFIG\_VFD

USIGN16	max_no_vfds	max. number of VFD's (1 .. MAX_VFD)	(1)
USIGN16	max_no_obj_descr	max. number of OD object descriptions	
USIGN8	max_obj_name_length	max. size of OD object name	
USIGN8	max_obj_ext_length	max. size of OD object extension	

### CRL-Configuration

#### Data structure T\_FMB\_CONFIG\_CRL

USIGN16	max_no_fal_sdb	max. number of FAL service description blocks	
USIGN16	max_no_fdl_sdb	max. number of FDL service description blocks	
USIGN16	max_no_data_buffer	max. number of PDU buffers	
USIGN16	max_no_api_buffer	max number of abort/poll/idle PDU buffers	
USIGN16	max_no_poll_entries	max number of poll list entries	
USIGN16	max_no_subscr_entries	reserved	
PB_BOOL	resrc_check	with resource check	(1)
USIGN8	max_no_parallel_req	max number of parallel services to send	(1)
USIGN8	max_no_parallel_ind	max number of parallel services to receive	(1)
USIGN8	dummy	alignment byte	

### DP-Configuration

#### Data structure T\_FMB\_CONFIG\_DP

USIGN8	max_number_slaves	maximum number of supported DP Slaves	
USIGN8	max_slave_output_len	max. length of slave output data per DP Slave	
USIGN8	max_slave_input_len	max. length of slave input data per DP Slave	
USIGN8	max_slave_diag_len	max. length of one diagnostic buffer entry	
USIGN16	max_slave_diag_entries	max. number entries in circular diagnostic buffer	
USIGN16	max_bus_para_len	max. length of DP Master bus parameter set	
USIGN16	max_slave_para_len	max. length of one DP Slave parameter set	

### FDLIF-Configuration

#### Data structure T\_FMB\_CONFIG\_FDLIF

USIGN8	send_req_credits	max number of send credits for SDA and SDN services	
USIGN8	srd_req_credits	max number of send credits for SRD services	
USIGN8	receive_credits	max number of receive credits	
USIGN8	max_no_resp_saps	max number of responder saps	

### SM7-Configuration

#### Data structure T\_FMB\_CONFIG\_SM7

(1)

USIGN16	reserved	reserved for future use
---------	----------	-------------------------

(1) for future use

**Notes on the data structures of the FMB-Set-Configuration service:**
**T\_FMB\_SET\_CONFIGURATION\_REQ**

With the configuration flags *fms\_active*, *dp\_active* and *fdlif\_active* the PROFIBUS user specifies which protocol components are active.

With the flag *fdl\_evt\_receiver* the user decides which user layer will receive FM2 events. (FM2 is the management instance of FDL). The user can select FMB\_USR, FM7\_USR, DP\_USR or FDLIF\_USR as event receiver. Only a subset of layer identifiers may be valid if not all configuration flags are set to PB\_TRUE. The following table shows which layer identifiers are valid in depending on the configuration flags.

configuration flag	valid layer identifier
<i>fms_active</i> = PB_TRUE	FMB_USR, FM7_USR
<i>dp_active</i> = PB_TRUE:	FMB_USR, DP_USR
<i>fdlif_active</i> = PB_TRUE	FMB_USR, FDLIF_USR
<i>sm7_active</i> = PB_FALSE	for internal use

The component *data\_buffer\_length* describes the length of buffer that are used for sending messages to remote stations and for receiving messages from remote stations. If you are not sure what buffer length your application needs, set the value 0xFF for this component.

It depends on the configuration flags which substructures of T\_FMB\_CONFIGURATION\_REQ are of significance. The following table shows what substructures have to be filled in depending on the configuration flags:

configuration flag	substructures that has to be filled
<i>fms_active</i> = PB_TRUE:	T_FMB_CONFIG_VFD and T_FMB_CONFIG_CRL
<i>dp_active</i> = PB_TRUE:	T_FMB_CONFIG_DP
<i>fdlif_active</i> = PB_TRUE:	T_FMB_CONFIG_FDLIF

**T\_FMB\_CONFIG\_CRL**

The FMS user has to specify the memory requirements for a CRL in terms of *max\_no\_fal\_sdfs*, *max\_no\_fdl\_sdfs*, etc. The user can get proper values for this components in two ways. The first way is to calculate the memory requirements for a CRL by means of the formulars that are given in chapter 4. A more comfortable way to find out the memory requirements is offered by the User Toolkit. The Toolkit provides C-functions that do low level checks on a CRL and calculate the memory requirements. For detailed information see the manual of the User Toolkit.

### T\_FMB\_CONFIG\_DP

By means of this data structure the user application is allowed to determine the maximum memory requirements for the DP protocol stack. Once these values have been defined the internal DP configuration (e.g. number of loaded DP Slaves) may be changed dynamically within these maximum limits.

Note, some parameters are allowed to be set to zero. This is possible because a DP Master can be used as individual configuration device (DP Master class2). In this case the memory for the slave handling can be preserved.

The parameters have the following meanings and ranges:

#### **max\_number\_slaves (0..DP\_MAX\_NUMBER\_STATIONS / 0..127)**

- defines the maximum number of DP Slaves which the DP Master is able to manage
- the real number of used (i.e. downloaded) DP Slaves may differ between zero and *max\_number\_slaves* during a DP session

#### **max\_slave\_input\_len / max\_slave\_output\_len (Even USIGN16)**

- defines the maximum number of bytes that **one** DP Slave may use for input / output values within the shared memory (between the DP User application and the DP protocol software)
- since the DP protocol software supports different ways of administering the shared I / O memory (see *Address Assignment Modes* in the DP documentation) these parameters determine the size of the whole I / O memory: size = max\_number\_slave \* (max\_slave\_input\_len + max\_slave\_output\_len)
- both values must be even values because the starting offset for a DP Slave I / O area must be aligned (in Address Assignment Mode ARRAY)

#### **max\_slave\_diag\_entries (max\_number\_slaves..65535)**

- determines the maximum number of diagnostic elements which the DP Master stores in a circular buffer
- if more than *max\_slave\_diag\_entries* messages are available at a certain time the oldest diagnostic element will be lost (recommended are at least 2 buffers per DP Slave)

#### **max\_slave\_diag\_len (DP\_MIN\_SLAVE\_DIAG\_LEN..DP\_MAX\_SLAVE\_DIAG\_DATA\_LEN / 6..244)**

- defines the maximum size in bytes of each diagnostic element that is used in the circular buffer

#### **max\_bus\_para\_len (DP\_MIN\_BUS\_PARA\_LEN..USIGN16 / 66..65535)**

- determines the maximum length for the DP bus parameter set including all *master\_user\_data*
- this area is accessed via *area\_code* = 127 during *DP\_Download* services

**max\_slave\_para\_len (DP\_MIN\_SLAVE\_PARA\_LEN..USIGN16 / 32..65535)**

- determines the maximum length for the DP Slave parameter sets
- this area is accessed via *area\_code* = 0..DP\_DEFAULT\_SLAVE\_ADDRESS (126) during DP\_Download services

### Notes on data structure T\_FMB\_CONFIG\_FDLIF

In the substructure T\_FMB\_CONFIG\_FDLIF the user configures the number of resources that are available for FDLIF service calls.

The component *send\_req\_credits* specifies the maximum number of parallel SDA and/or SDN service executions. If, for example, the *send\_req\_credits* are five, the FDLIF user can issue five SDA- or SDN-requests without waiting for a SDA- or SDN-confirmation. If the user exceeds his credits the PROFIBUS send function returns with negative result (error code: E\_IF\_SERVICE\_CONSTR\_CONFLICT).

The component *srd\_req\_credits* specifies the maximum number of parallel SRD-execution.

The component *receive\_credits* specifies the number of resources that are available for receiving PDUs from remote stations. The receive resources are shared among the FDL SAPs. When activating a SAP the FDLIF user specifies how many receive resources belong to the SAP (component *credits* in T\_FDLIF\_ACTIVATE\_SAP\_REQ). The sum of credits given to all SAPs must not exceed *receive\_credits* specified in the configuration service.

The component *max\_no\_resp\_saps* specifies the number of FDL Responder SAPs (FDL RSAPs) that may be activated by the FDLIF user.

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_SET_CONFIGURATION
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### **3.2 FMB-Set-Value Services**

The PROFIBUS API provides two services to set the FDL operational parameters:

- FMB-Set-Busparameters
- FMB-Set-Value-Loc



### 3.2.1 FMB-Set-Busparameter

This service is used to set all FDL operational parameters that are necessary to start FDL. This set of operational parameters is designated FDL bus parameters.

The FMB-Set-Busparameter service is similar to the services FM7-Set-Busparameter and DP-Download-Loc. Which service the PROFIBUS user has to execute depends on the operation mode which was specified in the FMB-Set-Configuration request. If a mixed operation mode has been chosen, e.g. FMS and DP have been activated, the FDL bus parameters **must** be set by the FMB-Set-Busparameter service. If FMS or DP should run in stand alone mode, the FDL Bus Parameters can be set by the FM7-Set-Busparameter service or DP-Download-Loc service respectively.

Note, that the DP protocol stack has some more DP specific parameters within the bus parameter set (e.g. poll\_timeout, min\_slave\_intervall, etc.). In case of mixed operation the FMB-Set-Busparameter service sets the DP specific parameters with default values (see chapter 4.2.5). To change these values use the DP-Set-Busparameter service (see manual DP Services chapter 4.3.9).

The FMB-Set-Busparameter service must be started immediately after the execution of the FMB-Set-Configuration service.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_SET_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

Data structure	T_FMB_SET_BUSPARAMETER_REQ	
USIGN8	loc_add	local station address
USIGN8	loc_segm	local segment
USIGN8	baud_rate	baud rate
USIGN8	medium_red	medium redundancy
USIGN16	tsl	slot time
USIGN16	min_tsdr	min. station delay time resp.
USIGN16	max_tsdr	max. station delay time resp.
USIGN8	tqui	quiet time
USIGN8	tset	setup time
USIGN32	ttr	target token rotation time
USIGN8	g	gap update factor
PB_BOOL	in_ring_desired	active or passive station
USIGN8	hsa	highest station address in ring
USIGN8	max_retry_limit	max. retry limit in case of transmit error

The valid values of the FDL operational parameters are explained in detail in chapter 4.2 "FDL Bus Parameters".

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_SET_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 3.2.2 FMB-Set-Value

The Set-Value-Loc service is used to set a single FDL operational parameter.

*Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_SET_VALUE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

*Data block for Request:*

Data structure T\_FMB\_SET\_VALUE\_REQ

USIGN8	id	value identifier
USIGN8	length	length of value field
USIGN8	value[length]	value

Parameter identifiers:

FDL-operational parameters which can be changed:

ID_BAUD_RATE	2	Baud rate
ID_TSL	6	Slot-Time
ID_MIN_TSDR	7	Minimum Station Delay Time
ID_MAX_TSDR	8	Maximum Station Delay Time
ID_TQUI	9	Time out
ID_TSET	10	Setup Time
ID_TTR	11	Target Rotation Time
ID_G	12	GAP-Update-Factor
ID_MAX_RETRY_LIMIT	15	Max. # of repetitions in case of transmit error

The valid values of the FDL operational parameters are explained in detail in chapter 4.2 "FDL Bus Parameters".

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_SET_VALUE
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure T\_ERROR standard error structure

### **3.3 FMB-Read-Value Services**

The PROFIBUS API offers two services to read the FDL operational parameters:

- FMB-Read-Busparameters
- FMB-Read-Value

### 3.3.1 FMB-Read-Busparameter

The FMB-Read-Busparameters service is used to read the FDL bus parameters.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_READ_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

n/a

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_READ_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

Data structure	T_FMB_READ_BUSPARAMETER_CNF	
USIGN8	loc_add	local station address
USIGN8	loc_segm	local segment
USIGN8	baud_rate	baud rate
USIGN8	medium_red	medium redundancy
USIGN16	tsl	slot time
USIGN16	min_tsdr	min. station delay time resp.
USIGN16	max_tsdr	max. station delay time resp.
USIGN8	tqui	quiet time
USIGN8	tset	setup time
USIGN32	ttr	target token rotation time
USIGN8	g	gap update factor
PB_BOOL	in_ring_desired	active or passive station
USIGN8	hsa	highest station address
USIGN8	max_retry_limit	max. retry limit
USIGN16	reserved	not used
USIGN8	ident[202]	FDL ident string

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

The read values of the FDL operational parameters are explained in detail in chapter 4.2 "FDL Bus Parameters".

### 3.3.2 FMB-Read-Value

The FMB-Read-Value-Loc service enables the PROFIBUS user to read aa single FDL operational parameter.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_READ_VALUE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

Data structure                      T\_FMB\_READ\_VALUE\_REQ

USIGN8	id	value identifier
USIGN8	dummy	alignment byte

Parameter identifiers:

FDL operational parameters:

ID_TS	1	Station address
ID_BAUD_RATE	2	Baudrate
ID_MEDIUM_RED	3	Redundancy
ID_HW_RELEASE	4	Hardware release
ID_SW_RELEASE	5	Software release
ID_TSL	6	Slot-Time
ID_MIN_TSDR	7	Minimum Station Delay Time
ID_MAX_TSDR	8	Maximum Station Delay Time
ID_TQUI	9	Time out
ID_TSET	10	Setup Time
ID_TTR	11	Target Rotation Time
ID_G	12	GAP-Update-Factor
ID_IN_RING_DESIRED	13	in ring desired
ID_HSA	14	Highest station address in local segment
ID_MAX_RETRY_LIMIT	15	Max. # of repetitions in case of transmit error
ID_LAS	17	List of active stations (LAS) (1)

(1) LAS is available only for active stations (in\_ring\_desired==PB\_TRUE)

The read values of the FDL operational parameters are explained in detail in chapter 4.2 "FDL Bus Parameters".

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_READ_VALUE
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

Data structure                      T\_FMB\_READ\_VALUE\_CNF

USIGN8	id	value identifier
USIGN8	length	length of value field
USIGN8	value[length]	value

result = NEG:

Data structure                      T\_ERROR                      standard error structure

LAS coding:

length	1..126	number of active stations
value[0]	0..126	1st station address
value[1]	0..126	2nd station address
value[n]	0..126	last station address

### 3.4 FMB-LSAP-Status

The FMB-LSAP-Status service is used to request the configuration of a local FDL Service Access Point (SAP).

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_LSAP_STATUS
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

Data structure	T_FMB_LSAP_STATUS_REQ	
USIGN8	lsap	local sap
USIGN8	dummy	alignment byte

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_LSAP_STATUS
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

Data structure	T_FMB_LSAP_STATUS_CNF	
USIGN8	access	station address
USIGN8	addr_extension	segment number
USIGN8	sda	SDA service
USIGN8	sdn	SDN service
USIGN8	srd	SRD service
USIGN8	csrd	CSR service

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------



Service coding:

Each service field (*sda*, *sdn*, *srd*, *csrd*) contains the arithmetic sum of the parameters "service" and the "role\_in\_service":

service:

SDA_RESERVED	0x00	SDA service
SDN_RESERVED	0x01	SDN service
SRD_RESERVED	0x03	SRD service
CSRD_RESERVED	0x05	CSRD service

role\_in\_service:

INITIATOR	0x00	initiator role
RESPONDER	0x10	responder role
BOTH_ROLES	0x20	initiator / responder
SERVICE_NOT_ACTIVATED	0x30	service not activated

### 3.5 FMB-Get-Live-List

This service provides the FMB user with an up-to-date list of all stations that are functional on the bus. The service is not available for passive stations.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_GET_LIVE_LIST
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

n/a

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_GET_LIVE_LIST
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

Data structure	T_FMB_GET_LIVE_LIST_CNF	
USIGN8	dummy	alignment byte
USIGN8	no_of_elements	number of live list elements
T_FMB_LIVE_LIST	live_list[no_of_elements]	live list

result = NEG:

T_ERROR	error	standard error structure
---------	-------	--------------------------

## Basic Management

---

Data structure	T_FMB_LIVE_LIST	
USIGN8	station	station address (0..126)
USIGN8	status	current status of station
Stati:		
PASSIVE	0x00	passive station
ACTIVE_NOT_READY	0x01	active station, not ready
ACTIVE_READY	0x02	active station, ready to enter ring
ACTIVE_IN_RING	0x03	active station in ring

## 3.6 FMB-FM2-Event

With the FMB-FM2-EVENT service, the FMB indicates FM2 events to the PROFIBUS user.

In the FMB-SET-CONFIGURATION service the PROFIBUS user selects the instance that receives the FM2 events. If in the FMB-SET-CONFIGURATION request the flag *fdl\_evt\_receiver* was set to FMB\_USR, the FM2 events are indicated as FMB-FM2-Events.

*Service-Description-Block for die Indication:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USER
USIGN8	service	FMB_FM2_EVENT
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	not used

*Data Block:*

Data structure	T_FMB_FM2_EVENT_IND	
USIGN16	reason	event reason

**FM2 event reason codes:**

FM2_FAULT_ADDRESS	1	duplicate address recognized	
FM2_FAULT_PHY	2	physical layer is malfunctioning	(1)
FM2_FAULT_TTO	3	timeout on bus detected	
FM2_FAULT_SYN	4	no receiver synchronization	
FM2_FAULT_OUT_OF_RING	5	local station out of ring	
FM2_GAP_EVENT	6	GAP area has changed	(1)

(1) Not supported by ASPC2

**Additional FM2 event reason codes (Error messages from ASPC2)**

FM2_MAC_ERROR	19	fatal MAC error
FM2_HW_ERROR	20	fatal HW error

## 3.7 FMB-Reset

The FMB-Reset service is used to reset the whole communication software (FAL, DP, FDLIF and FDL).

### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_RESET
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### *Data block for Request:*

n/a

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_RESET
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### **Note:**

In future releases of SOFTING's PROFIBUS API, the service FMB-Reset will not be supported. The service is provided only for compability with former releases of PROFIBUS API. In new applications, use FMB-Exit instead of FMB-Reset.

## 3.8 FMB-Exit

This service terminates the PROFIBUS communication software (FAL, DP, FDLIF and FDL).

### *Service-Description-Block for Request:*

USIGN16	comm_ref	0
USIGN8	layer	FMB
USIGN8	service	FMB_EXIT
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### *Data block for Request:*

n/a

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_EXIT
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 3.9 FMB-Exception

A FMB-Exception indicates a fatal system error.

If a fatal system error occurs, please contact SOFTING for analysing the exception parameters.

### *Service-Description-Block for Indication:*

USIGN16	comm_ref	0
USIGN8	layer	FMB_USR
USIGN8	service	FMB_EXCEPTION
USIGN8	primitive	IND
INT8	invoke_id	0
INT16	result	POS

### *Data block for Indication:*

Data structure	T_EXCEPTION	
USIGN8	task_id	task in wich the fatal system error is occurred
USIGN8	par1	exception parameter 1
USIGN16	par2	exception parameter 2
USIGN16	par3	exception parameter 2

## 4 CONFIGURATION PARAMETERS

### 4.1 FMB Configuration

The FMB-Set-Configuration service determines the size of RAM that is dynamically allocated by the PROFIBUS protocol stack. The following chapter gives an overview of the memory requirements for FAL, DP and FDLIF. The following formulas calculate net sizes - the management overhead of the operating system is not considered.

#### 4.1.1 VFD Configuration

##### Memory requirements for VFDs:

The component *vfd.max\_no\_vfds* in the FMB-SET-CONFIGURATION request specifies the number of VFDs that are supported by FAL. The memory requirement for one VFD is:

$$(76 + 3 \cdot \text{VFD\_STRING\_LENGTH}^{(1)}) \text{ bytes}$$

- (1) The constant VFD\_STRING\_LENGTH is defined in include file PB\_CONF.H. It is not changeable by the user!

##### Memory requirements for Object Dictionaries:

Memory to store communication objects is allocated while loading the Object Dictionary (OD). At creation time of the OD header the total size of the Object Dictionary is calculated and the memory for the Object Descriptions is allocated.

Memory requirements for one Object Description:

$$(18 + \text{od\_obj\_descr.length}^{(2)} + \text{vfd.max\_obj\_ext\_length}^{(3)}) \text{ bytes}$$

- (2) The name length of a Object Description is fixed by loading the OD header (see FMS manual).
- (3) The extension length of a Object Description is fixed by the configuration parameter *vfd.max\_obj\_ext\_length* ( see FMB\_SET\_CONFIGURATION service).



### 4.1.2 CRL Configuration

The memory requirements of a CRL depends on the number of CRL entries, the connection types, the number of parallel services, etc. Below you will find formulas to calculate the memory requirements for a Communication Relationship (CR) in dependence on connection type, parallel services, etc.

The CRL handles with several buffer types. There are Fieldbus Application Layer Service Description Blocks (FAL-SDBs), Fieldbus Data link Layer Service Description Blocks (FDL-SDBs), PDU buffer (Data-Blocks), Abort/Poll/Idle-buffer (API-Blocks) and Poll-List-Entries. The sizes of the buffers are as follows

FAL-SDB	42 bytes
FDL-SDB	36 bytes
Data-Block	50 to 255 bytes
API-Data-Block	30 bytes
Poll-List-Entry	18 bytes

The formulas contain some abbreviations with the following meaning:

scc	Send confirmed request counter in CRL entry
sac	Send acknowledge request counter in CRL entry
rcc	Receive confirmed request counter in CRL entry
rac	Receive acknowledge request counter in CRL entry
ci	Control interval in CRL entry
mult	Multiplier in CRL entry

#### 4.1.2.1 Buffers for a Master-Master CR

The connection type of a Master-Master CR is MMAC. The following formulas are used to calculate the buffer requirements for one Master-Master CR:

<b>Number of FAL-SDBs:</b>	$\max(1, scc+sac) + \max(1, rcc+rac) + \min(ci, 1) + 3$
<b>Number of FDL-SDBs:</b>	$2*\max(1, scc+sac) + \max(1, rcc+rac) + 2*\min(ci, 1) + 4$
<b>Number of Data-Blocks:</b>	$2*\max(1, scc+sac) + \max(1, rcc+rac) + \min(ci, 1) + 2$
<b>Number of API-Blocks:</b>	$\min(ci, 1) + 1$

By using the connection attribute I\_CONN for Master-Master CRs the buffer requirements can be reduced.

With this connection attribute the user can build groups of connections where all members of a group use the same FDL SAP (see component *loc\_isap* the static part of the CRL entry). It is possible to build such a group on condition that only one member of the group is in data transfer phase at one point in time and all members act as initiate requester. Within an I\_CONN-group, the resources are shared.

An example: Let say, there are three master/master connections with connection attribut I\_CONN, all using the same FDL SAP. The connections may have the following demand for resources:

	FAL-SDBs	FDL-SDBs	Data-Blocks	API-Blocks
Connection A	10	15	13	2
Connection B	12	17	15	2
Connection C	8	11	10	1

The demand of resources for the group is built as maximum of each column. So, for this example the result is:

	FAL-SDBs	FDL-SDBs	Data-Blocks	API-Blocks
Group	12	17	15	2

#### 4.1.2.2 Buffers for Masters in Master-Slave CRs

Master-Slave CRs have the connection type MSAC, MSAC\_SI, MSCY or MSCY\_SI. The component *loc\_sap* in the static part of the CRL entry determines the role the station plays in that CR. If *loc\_sap* is equal to *poll\_sap* (*poll\_sap* is member of the CRL header), the station acts as Master otherwise it acts as Slave.

##### Buffers for a Master in an acyclic Master-Slave CR (connection type MSAC and MSAC\_SI)

Number of FAL-SDBs:  $\max(1, scc + sac) + rac + \min(1, ci) + 3$   
 Number of FDL-SDBs:  $sac + rac + 1$   
 Number of Data-Blocks:  $\max(1, scc + sac) + sac + rac + 3$   
 Number of API-Blocks:  $rac + \min(1, ci) + 2$   
 Number of Poll-List-Entries: 1

##### Buffers for a Master in a cyclic Master-Slave CR (connection type MSCY and MSCY\_SI)

Number of FAL-SDBs:  $sac + rac + 4$  (\*)  
 Number of FDL-SDBs:  $sac + rac + \max(1, mult)$  (\*)  
 Number of Data-Blocks:  $2 * sac + rac + \max(1, mult) + 4$  (\*)  
 Number of API-Blocks:  $rac + \max(1, mult) + 1$   
 Number of Poll-List-Entries:  $\max(1, mult)$

(\*) For the first Master-Slave CR in the CRL the FDL-SDB resources have to be incremented by 1. This resource is used for the *poll\_sap*.

(\*) If, in the static part of a CRL entry in the component 'conn\_type' the LLI event bit is set, the FAL- and DATA resources have to be incremented by 1.

#### 4.1.2.3 Buffers for Slaves in Master-Slave CRs

Master-Slave CRs have the connection type MSAC, MSAC\_SI, MSCY or MSCY\_SI. The component *loc\_isap* in the static part of the CRL entry determines the role the station plays in that CR. If *loc\_isap* is not equal to *poll\_sap* (*poll\_sap* is member of the CRL header), the station acts as Slave otherwise it acts as Master.

##### Buffers for a Slave in an acyclic Master-Slave CR (connection type MSAC and MSAC\_SI)

Number of FAL-SDBs:	$sac + \max(1, rcc + rac) + \min(1, ci) + 3$
Number of FDL-SDBs:	$2 * sac + \max(1, rcc + rac) + 2 * \min(1, ci) + 7$
Number of Data-Blocks:	$2 * sac + \max(1, rcc + rac) + 2 * \min(1, ci) + 6$

##### Buffers requirements for a Slave in a cyclic Master-Slave CR (connection type MSCY and MSCY\_SI)

Number of FAL-SDBs:	$sac + rac + 4$
Number of FDL-SDBs:	$2 * sac + rac + 8$
Number of Data-Blocks:	$2 * sac + rac + 8$

#### 4.1.2.4 Buffers for connectionless CRs

Connectionless CRs have the connection type MULT or BRCT. The component *rem\_add* in the static part of the CRL entry determines the role the station plays in that CR. If *rem\_add* is equal to 127 the station acts as Sender otherwise it acts as Receiver.

##### Buffer requirements for multicast/broadcast Sender

Number of FAL-SDBs:	$\max(1, sac)$
Number of FDL-SDBs:	$\max(1, sac) + 1$
Number of Data-Blocks:	$\max(1, sac)$

##### Buffer requirements for the multicast/broadcast Receiver

Number of FAL-SDBs:	$\max(1, rac)$
Number of FDL-SDBs:	$\max(1, rac) + 1$
Number of Data-Blocks:	$\max(1, rac)$

### 4.1.3 DP Configuration

The DP protocol software uses a definable size of memory that is not changeable after issuing the FMB-Set-Configuration service. Afterwards this static amount of memory blocks is distributed "dynamically" during a DP session (e.g. adding and deleting of DP Slaves, etc.).

The amount of memory that is used by the DP protocol software can be calculated as follows:

Memory Type	Block Size (Bytes)	Number Blocks
bus parameter set	T_FMB_CONFIG_DP::max_bus_para_len	2
slave parameter set	T_FMB_CONFIG_DP::max_slave_para_len	T_FMB_CONFIG_DP::max_number_slaves + 1
circular diagnostic buffer	T_FMB_CONFIG_DP::max_slave_diag_len	T_FMB_CONFIG_DP::max_slave_diag_entries
I/O data memory (beside DPRAM)	T_FMB_CONFIG_DP::max_slave_input_len + T_FMB_CONFIG_DP::max_slave_output_len	1 if T_FMB_CONFIG_DP::max_number_slaves
internal DP memory	34	T_FMB_CONFIG_DP::max_number_slaves + 6

### 4.1.4 FDLIF Configuration

FDLIF uses FDL-SDBs and Data-Blocks (see CRL configuration). The following table gives an overview of the relation between FDLIF credits and buffer requirements:

One credit	occupies FDL-SDB	occupies Data-Block(s)
SDA/SDN credit	1	1
SRD credit	1	2
receive credit	1	1
FDL RSAP	1	0

#### 4.1.5 Standard Configuration

If the PROFIBUS user starts with any other service (e.g. FMB-Set-Busparameter), FMB assumes a standard configuration. In this standard configuration only FAL is active.

The following table shows the values of the standard configuration for PROFIboard and PROFIcard controller:

Configuration Parameters	PROFIBUS Controller		Description
	PROFIboard PROFI104	PROFIcard	
<b>VFD-Configuration</b>			
max_no_vfds	5	5	max. number of VFD's
max_no_obj_descr	823	823	max. number of OD object descriptions
max_obj_name_length	32	32	max. size of OD object name
max_obj_ext_length	32	32	max. size of OD object extension
<b>CRL-Configuration</b>			
max_no_fal_sdb	390	200	max. number of FAL service description blocks
max_no_fdl_sdb	580	230	max. number of FDL service description blocks
max_no_data_buffer	440	230	max. number of PDU buffers
max_no_api_buffer	100	50	max number of abort/poll/idle PDU buffers
max_no_poll_entries	48	48	max number of poll list entries
max_no_subscr_entries	0	0	reserved

## 4.2 FDL Bus Parameters

### 4.2.1 Range of Values

The following figure lists all Bus Parameters with their definition and the range of value.

PARAMETER	RANGE OF VALUE	DESCRIPTION
loc_add	0 .. 126	local station address
loc_segm	0 .. 63 255	local segment address no segment address
baud_rate	0 1 11 2 3 4 6 7 8 9	9.6 Kbit/s 19.2 Kbit/s 45.45 Kbit/s 93.75 Kbit/s 187.5 Kbit/s 500 Kbit/s 1.5 Mbit/s 3 Mbit/s 6 Mbit/s 12 Mbit/s
medium_red	0	no bus redundancy
tsl	37 .. 16383	slot time
min_tsdr	11 .. 1023	min. station delay time
max_tsdr	37 .. $2^{16}-1$	max station delay time
tqui	0 .. 493	quiet time
tset	1 .. 494 - tqui	setup time
ttr	256 .. $2^{24}-1$	target rotation time
g	1 .. 100	gap update factor
in_ring_desired	0 255	passive station active station
hsa	1 .. 126	highest station address
max_retry_limit	0 .. 7	max retry limit

Description of the bus parameters in detail:

**loc\_add**                      local station address  
0 .. 126                      This parameter defines the local station address.

**baud\_rate**                      baud rate

0	9,6	Kbit/s
1	19,2	Kbit/s
11	45,45	Kbit/s
2	93,75	Kbit/s
3	187,5	Kbit/s
4	500	Kbit/s
6	1,5	Mbit/s
7	3	Mbit/s
8	6	Mbit/s
9	12	Mbit/s

This parameter defines the baud rate.

**medium\_red**                      medium redundancy  
0                              single, no redundancy

**tsl**                              slot time  
37..16383                      [Bit Times]

The slot time is the maximum time a master station must wait for a transaction response.

**Note:** The selected slot time must always be greater than  $\text{max\_TSDR} + \text{TQUI} + 11 + 2$ .

**min\_tsdr**                      minimum station delay time for responder  
11 .. 1023                      [Bit Times]

The min\_tsdr is the period of time which may elapse before the responder can send the response frame.

**max\_tsdr**                      maximum station delay time for responder  
37 ..  $2^{16}-1$                       [Bit Times]

The responder has to send the response frame before the max\_tsdr is elapsed.

**t\_qui**                      quiet time  
0 .. 493                  [Bit Times]

The quiet time is the period of time which a transmitting station must wait after the end of a frame before enabling its receiver. It is significant when using repeaters.

**t\_set**                      setup time  
1 .. 494 - t\_qui        [Bit Times]

The setup time is the time between the occurrence of an interrupt request and the necessary request action is performed.

**ttr**                          target rotation time  
256 ..  $2^{24}-1$         [Bit Times]

The target rotation time is the anticipated time for one token rotations on the PROFIBUS network, including allowances for high and low priority transactions and GAP maintenance.

**g**                            gap update factor  
1 .. 100

This parameter defines the number of token rotations between GAP maintenance cycles.

**in\_ring\_desired**        in ring desired  
0                          passive station  
255                        active station

This parameter defines whether the station is a active (master) or an passive (slave) station.

**hsa**                        highest station address  
1 .. 126

This parameter defines the highest station address. The upper limit is determined by the PROFIBUS protocol. For passive stations the HSA has no significance.

**max\_retry\_limit**        maximum retry limit  
0 .. 7

This parameter max\_retry\_limit indicates how often FDL has to repeat the request frame when no response or acknowledgement frame is received from the recognized station within the slot time.



#### 4.2.2 Recommended Bus Parameters for FMS Operation

For FMS operation the **PNO** recommends default values for baudrates up to 500 kbit/s. The values up to 12000 kbit/s are recommend by EN 50170 / 2 (DP) and by the implementation guides to EN E 50170 / 2 (DP). The default values for 45,45 kbit/s are not defined for single FMS operation.

Parameter/Baudrate	9,6 KBaud	19,2 KBaud	45,45 KBaud	93.75 Kbaud	187,5 KBaud	500 KBaud	1,5 MBaud	3 MBaud	6 MBaud	12 MBaud
T <sub>SL</sub> [bit times]	100	200	-	500	1000	2000	3000	400	600	1000
min_T <sub>SDR</sub> [bit times]	30	60	-	125	250	500	150	11	11	11
max_T <sub>SDR</sub> [bit times]	50	100	-	250	500	1000	980	250	450	800
T <sub>SET</sub> [bit times]	5	10	-	15	25	50	240	4	8	16
T <sub>QUI</sub> [bit times]	22	22	-	22	22	22	0	3	6	9
G	1	1	-	1	1	1	10	10	10	10
HSA	126	126	-	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	-	1	1	1	1	2	3	4
T <sub>TR</sub> [bit times]	10000	15000	-	30000	50000	100000	300000	600000	1200000	2400000

### 4.2.3 Recommended Bus Parameters for FMS Operation using ASPC2

For FMS operation using ASIC ASPC2 use the default values for baudrates up to 12000 kbit/s recommend by EN 50170 /2 (DP) and by the implementation guides to EN E 50170 / 2 (DP). The default values for 45,45 kbit/s are not defined for single FMS operation.

Parameter/Baudrate	9,6 KBaud	19,2 KBaud	45,45 Kbaud	93.75 Kbaud	187,5 KBaud	500 KBaud	1,5 MBaud	3 MBaud	6 MBaud	12 MBaud
T <sub>SL</sub> [bit times]	100	200	-	500	1000	2000	3000	400	600	1000
min_T <sub>SDR</sub> [bit times]	30	60	-	125	250	500	150	11	11	11
max_T <sub>SDR</sub> [bit times]	50	100	-	250	500	1000	980	250	450	800
T <sub>SET</sub> [bit times]	1	1	-	1	1	1	240	4	8	16
T <sub>QUI</sub> [bit times]	0	0	-	0	0	0	0	3	6	9
G	1	1	-	1	1	1	10	10	10	10
HSA	126	126	-	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	-	1	1	1	1	2	3	4
T <sub>TR</sub> [bit times]	10000	15000	-	30000	50000	100000	300000	600000	1200000	2400000

#### 4.2.4 Recommended Bus Parameters for DP and FMS Operation

For mixed DP/FMS operation EN 50170/2 (DP) recommends baudrates up to 1500 kbit/s. The values for baudrates up to 12000 kbit/s are recommended by EN 50170 /2 (DP) and by the implementation guides to EN 50170 / 2 (DP).

The baudrate 45,45 kbit/s is only used for DP- and PA -systems with coupling devices.

Parameter/Baudrate	9,6 KBaud	19,2 KBaud	45,45 KBaud	93.75 Kbaud	187,5 KBaud	500 Kbaud	1,5 Mbaud	3 MBaud	6 MBaud	12 MBaud
T <sub>SL</sub> [bit times]	125	250	640	600	1500	3500	3000	400	600	1000
min_T <sub>SDR</sub> [bit times]	30	60	11	125	250	500	150	11	11	11
max_T <sub>SDR</sub> [bit times]	60	120	400	250	500	1000	980	250	450	800
T <sub>SET</sub> [bit times]	1	1	95	1	1	1	240	4	8	16
T <sub>QUI</sub> [bit times]	0	0	0	0	0	0	0	3	6	9
G	1	1	10	1	1	1	10	10	10	10
HSA	126	126	126	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	1	1	1	1	1	2	3	4
T <sub>TR</sub> [Bitzeiten]	-	-	-	-	-	-	-	-	-	-

### 4.2.5 Recommended Bus Parameters for DP Operation

For DP operation EN 50170/2 (DP) recommends baudrates up to 1500 kbit/s. The values up to 12000 kbit/s are recommend by and by the implementation guides to EN E 50170 / 2 (DP).

The baudrate 45,45 kbit/s is only used for DP- and PA -systems with coupling devices.

Parameter/Baudrate	9,6 KBaud	19,2 KBaud	45,45 KBaud	93.75 Kbaud	187,5 KBaud	500 KBaud	1,5 MBaud	3 MBaud	6 MBaud	12 MBaud
T <sub>SL</sub> [bit times]	100	100	640	100	100	200	300	400	600	1000
min_T <sub>SDR</sub> [bit times]	11	11	11	11	11	11	11	11	11	11
max_T <sub>SDR</sub> [bit times]	60	60	400	60	60	100	150	250	450	800
T <sub>SET</sub> [bit times]	1	1	95	1	1	1	1	4	8	16
T <sub>QUI</sub> [bit times]	0	0	0	0	0	0	0	3	6	9
G	1	1	10	1	1	1	10	10	10	10
HSA	126	126	126	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	1	1	1	1	1	2	3	4
T <sub>TR</sub> [Bitzeiten]	-	-	-	-	-	-	-	-	-	-

DP Busparameter default values:

Parameter	Default value	Meaning
bp_flag	0x00	bus parameter flag: error_action = FALSE (autoclear function disabled)
min_slave_interval	1 [100µs] = 100µs	data exchange cycles of the master as quick as possible
poll_timeout	1000 [1ms] = 1s	watchdog timer for Master / Master communication
data_control_time	100 [10ms] = 1s	control interval for Global_Control status reports of the DP Master
master_class2_name	empty	vendor name of the master in Master / Master communication (no used)
master_user_data_len	32 + 2 = 34	no master_user_data; this field contains the length of the vendor name and the length field itself

## APPENDIX A

## STANDARD ERROR STRUCTURE AND ERROR CODES

For negative confirmations the following standard error structure is used:

Data structure:	T_ERROR	
USIGN16	class_code	error class and error code
INT16	add_detail	additional detail
STRINGV	add_description[MAX_ERROR_DESCR_LENGTH]	additional description

The 16 bit *class\_code* parameter contains the error class in the high byte and the error code in the low byte.

The FMB uses the following codes for negative confirmations. Each pair of error class and error code is encoded in a constant. The 16 bit component *class\_code* in T\_ERROR contains the error class in the high byte and the error code in the low byte.

	class_code	error class	error code	Description Class	Code
E_FMB_RESOURCE_OTHER	0x0200	2	0	Resource	Other
E_FMB_RESOURCE_MEM_UNAVAILABLE	0x0201	2	1		Memory-Unavailable
E_FMB_SERV_OTHER	0x0300	3	0	Service	Other
E_FMB_SERV_OBJ_STATE_CONFLICT	0x0301	3	1		Object-State-Conflict
E_FMB_SERV_OBJ_CONSTR_CONFLICT	0x0302	3	2		Object-Constraint-Conflict
E_FMB_SERV_PARAM_INCONSIST	0x0303	3	3		Parameter-Inconsistent
E_FMB_SERV_ILLEGAL_PARAM	0x0304	3	4		Illegal-Parameter
E_FMB_SERV_PERM_INTERN_FAULT	0x0305	3	5		Permanent-Internal-Fault
E_FMB_ACCESS_OTHER	0x0500	5	0	Access	Other
E_FMB_ACCESS_OBJ_ACC_UNSUP	0x0501	5	1		Object-Access-Unsupported
E_FMB_ACCESS_OBJ_NON_EXIST	0x0502	5	2		Object-Non_Existent
E_FMB_ACCESS_OBJ_ACCESS_DENIED	0x0503	5	3		Object-Access-Denied
E_FMB_ACCESS_HARDWARE_FAULT	0x0504	5	4		Hardware-Fault
E_FMB_ACCESS_TYPE_CONFLICT	0x0505	5	5		Type-Conflict
E_FMB_OTHER	0x0700	7	0	Other	Other
E_FMB_CFG_DP_TOO_MANY_SLAVES	0x0901	9	1	Configuration	Too many slaves configured
E_FMB_CFG_DP_WRONG_IO_DATA_LEN	0x0902	9	2		I/O data length inconsistent
E_FMB_CFG_DP_IO_ALIGNMENT_ERROR	0x0903	9	3		odd I/O data length detected
E_FMB_CFG_DP_TOO_FEW_DIAG_ENTRIES	0x0904	9	4		not enough diagnostic buffers
E_FMB_CFG_DP_WRONG_DIAG_DATA_LEN	0x0905	9	5		invalid diag buffer length
E_FMB_CFG_DP_WRONG_BUS_PARA_LEN	0x0906	9	6		invalid bus parameter length
E_FMB_CFG_DP_WRONG_SLAVE_PARA_LEN	0x0907	9	7		invalid slave parameter length
E_FMB_CFG_DP_DPRAM_ERROR	0x0908	9	8		DP-RAM error



# **PROFIBUS Application Program Interface**

## **FMS Services**

Version 5.2  
Rev. 01

Date: 03-March-1998

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---



---

## CONTENTS

1 SCOPE .....	1
2 OVERVIEW .....	2
3 FMS CONTEXT MANAGEMENT SERVICES .....	6
3.1 INITIATE .....	6
3.2 ABORT .....	8
3.3 REJECT .....	11
4 VFD SUPPORT SERVICES .....	12
4.1 LOCAL SERVICES .....	12
4.1.1 Create-VFD .....	12
4.1.2 VFD-Set-Physical-Status .....	14
4.2 REMOTE SERVICES .....	15
4.2.1 Status .....	15
4.2.2 Unsolicited-Status .....	16
4.2.3 Identify .....	17
5 OD MANAGEMENT .....	18
5.1 LOCAL SERVICES .....	18
5.1.1 Loading an Object Dictionary .....	18
5.1.1.1 Initiate-Load-OD-LOC .....	18
5.1.1.2 Load-OD-LOC .....	20
5.1.1.3 Terminate-Load-OD-LOC .....	22
5.1.2 Reading an Object Description .....	23
5.1.2.1 OD-Read .....	23
5.2 REMOTE SERVICES .....	25
5.2.1 Get-OD .....	25
5.2.2 Put-OD Services .....	27
5.2.2.1 Initiate-Put-OD .....	27
5.2.2.2 Put-OD .....	29
5.2.2.3 Terminate-Put-OD .....	30
5.3 COMMUNICATION OBJECT STRUCTURE .....	31
5.3.1 Objects in Local Object Dictionary .....	31
5.3.2 PROFIBUS Object Description Transfer .....	39

6	VARIABLE ACCESS .....	44
6.1	READ.....	44
6.2	WRITE .....	46
6.3	READ-WITH-TYPE.....	48
6.4	WRITE-WITH-TYPE .....	50
6.5	INFORMATION-REPORT .....	51
6.6	INFORMATION-REPORT-WITH-TYPE .....	52
6.7	PHYSICAL-READ.....	53
6.8	PHYSICAL-WRITE .....	54
6.9	DEFINE-VARIABLE-LIST.....	55
6.10	DELETE-VARIABLE-LIST .....	57
6.11	VARIABLE-DATA-EVENT .....	58
7	DOMAIN-MANAGEMENT SERVICES .....	59
7.1	DOWNLOAD SERVICES .....	59
7.1.1	Initiate-Download-Sequence .....	59
7.1.2	Download-Segment.....	61
7.1.3	Terminate-Download-Sequence .....	62
7.1.4	Request-Domain-Download .....	63
7.2	UPLOAD SERVICES.....	64
7.2.1	Initiate-Upload-Sequence .....	64
7.2.2	Upload-Segment .....	66
7.2.3	Terminate-Upload-Sequence .....	67
7.2.4	Request-Domain-Upload .....	68
7.3	GENERIC-DOWNLOAD SERVICES .....	69
7.3.1	Generic-Initiate-Download-Sequence .....	69
7.3.2	Generic-Download-Segment.....	71
7.3.3	Generic-Terminate-Download-Sequence .....	72
8	PROGRAM-INVOCATION-MANAGEMENT SERVICES .....	73
8.1	CREATE-PROGRAM-INVOCATION .....	73
8.2	DELETE-PROGRAM-INVOCATION .....	75
8.3	START-PROGRAM-INVOCATION .....	77
8.4	STOP-PROGRAM-INVOCATION .....	78
8.5	RESUME-PROGRAM-INVOCATION .....	79
8.6	RESET-PROGRAM-INVOCATION .....	80
8.7	KILL-PROGRAM-INVOCATION.....	81
8.8	PI-SET-STATE-LOC.....	82

---

9 EVENT-MANAGEMENT SERVICES .....	84
9.1 EVENT-NOTIFICATION.....	84
9.2 EVENT-NOTIFICATION-WITH-TYPE .....	85
9.3 ACKNOWLEDGE-EVENT-NOTIFICATION.....	87
9.4 ALTER-EVENT-CONDITION-MONITORING .....	88
 APPENDIX A .....	 89
 ERROR STRUCTURE AND ERROR CODES.....	 89
 INDEX.....	 90



## 1 SCOPE

This manual describes the service-specific parameters and data for local and remote FMS services.

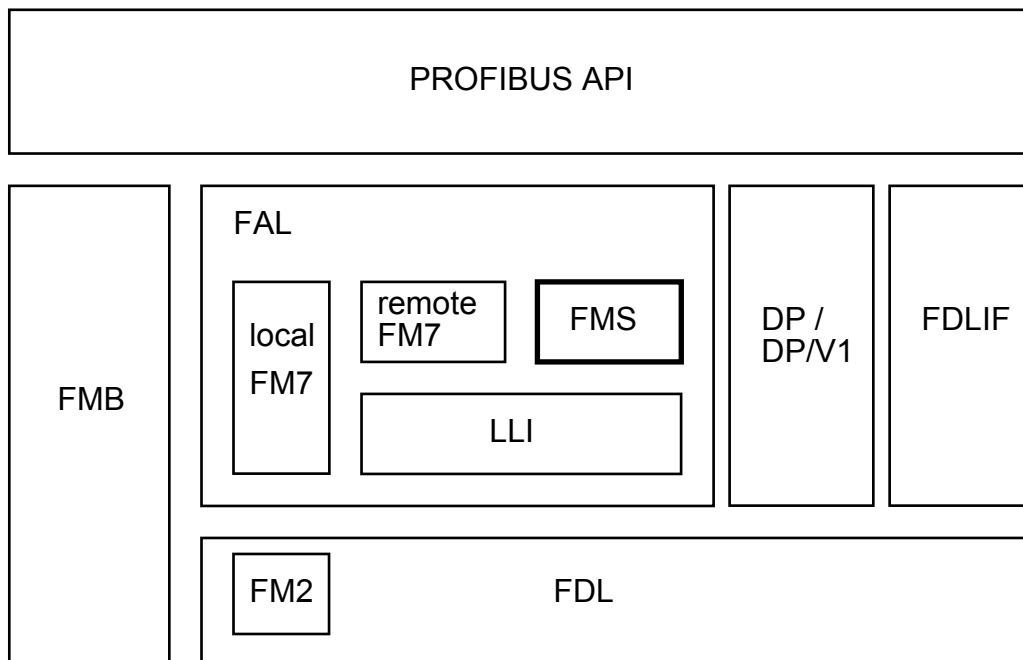
The Fieldbus Message Specification (FMS) services provide remote access to communication objects like variables, events, Domains and Program Invocations.

FMS is part of the Fieldbus Application Layer (FAL).

Softing's PROFIBUS Application Program Interface provides uniform access to all service groups of the PROFIBUS protocol. The common access functions are described in the "User Interface" part of the PROFIBUS User Manual.

This document describes the specific constants and data structures of all FMS services.

The FMS-specific types and constants are defined in the include file PB\_FMS.H.



This document should be read in conjunction with the following parts of the PROFIBUS User Manual:

- "User Interface" (describes the uniform access functions to all PROFIBUS services)
- "Basic Management" (describes the management services common to all protocol components)
- "FM7 Services" (describes the management services which are necessary to configure FAL)

## 2 OVERVIEW

The Fieldbus Message Specification (FMS) is based on a series of object oriented models that provide distributed applications with the functionality and services necessary for their implementation in an open, interoperable environment.

The service descriptions in this manual are grouped according to the FMS models:

**Context Management** services are used to establish a connection, to release a connection and to reject improper services.

The **Virtual Field Device (VFD)** is an abstract model for the description of the data and the behavior of an automation system as seen by a communication partner. A PROFIBUS device may contain one or more VFDs. FMS provides services to identify a VFD and to read or report the status of a VFD.

The **Object Dictionary (OD) Management** provides means to maintain the Object Dictionary, which contains descriptions of all communication objects.

**Variable Access** is probably the most commonly used model of FMS, as it provides services to read and write variable objects like simple variables, records, arrays and variable lists.

A Domain is a part of memory which may contain programs or data. The **Domain Management** provides services to download or upload Domains.

The **Program Invocation** model provides services to link Domains to a program, to start this program, to stop and delete it. More than one Program Invocation may be created in a device.

The **Event Management** model offers service to report and acknowledge events.

## FMS Services (in service group order)

Service group	Identifier	Code	Page
Context-Management	FMS_INITIATE	0	6
	FMS_ABORT	38	8
	FMS_REJECT	39	11
VFD local	FMS_CREATE_VFD_LOC	46	12
	FMS_VFD_SET_PHYS_STATUS_LOC	47	14
VFD remote	FMS_STATUS	2	15
	FMS_UNSOLICITEDSTATUS	34	16
	FMS_IDENTIFY	3	17
OD local	FMS_INIT_LOAD_OD_LOC	43	18
	FMS_LOAD_OD_LOC	44	20
	FMS_TERM_LOAD_OD_LOC	45	22
	FMS_OD_READ_LOC	42	23
OD remote	FMS_INIT_PUT_OD	30	27
	FMS_PUT_OD	31	29
	FMS_TERM_PUT_OD	32	30
	FMS_GET_OD	6	25
Variable service	FMS_READ	4	44
	FMS_READ_WITH_TYPE	7	48
	FMS_WRITE	5	46
	FMS_WRITE_WITH_TYPE	8	50
	FMS_INFO_RPT	33	51
	FMS_INFO_RPT_WITH_TYPE	36	52
	FMS_DEF_VAR_LIST	9	55
	FMS_DEL_VAR_LIST	10	57
	FMS_PHYS_READ	28	53
	FMS_PHYS_WRITE	29	54
	FMS_VAR_DATA_EVENT_LOC	49	58
Domain Download	FMS_INIT_DOWNL_SEQ	11	59
	FMS_DOWNL_SEG	12	61
	FMS_TERM_DOWNL_SEQ	13	62
	FMS_REQ_DOM_DOWNL	17	63
Generic Domain Download	FMS_GEN_INIT_DOWNL_SEQ	61	69
	FMS_GEN_DOWNL_SEG	62	71
	FMS_GEN_TERM_DOWNL_SEQ	63	72
Domain Upload	FMS_INIT_UPL_SEQ	14	64
	FMS_UPL_SEG	15	67
	FMS_TERM_UPL_SEQ	16	68
	FMS_REQ_DOM_UPL	18	69
Program Invocation	FMS_PI_CREATE	19	73
	FMS_PI_DEL	20	75
	FMS_PI_START	21	77
	FMS_PI_STOP	22	78
	FMS_PI_RESUME	23	79
	FMS_PI_RESET	24	80
	FMS_PI_KILL	25	81
	FMS_PI_SET_STATE_LOC	48	82
Event-Service	FMS_EVN_NOTIFY	35	84
	FMS_EVN_NOTIFY_WITH_TYPE	37	85
	FMS_ACK_EVN_NOTIFY	27	87
	FMS_ALT_EVN_CND_MNT	26	88





## Notes on Data Structures and Parameters

The FMS-specific types and constants are defined in the include file PB\_FMS.H.

All words, long-words, strings, arrays and records begin on even addresses. To accomplish this, fill bytes had to be added in some places. They are always recognizable by the name *dummy*.

Data blocks do not contain pointers. If a data block contains one or more fields or lists of variable length, then the length information of all variable-length fields is stored in the constant part. The fields of variable length follow on the constant part.

Here is an example of such a data block:

<b>constant parameters</b>
<b>length of field 1</b>
<b>length of field 2</b>
<b>field 1</b>
<b>field 2</b>

The variable data fields are shown between comment delimiters in the include file PB\_FMS.H to show their position and structure, without forcing the programmer to use data structures of a specific length. Nevertheless, the data must be entered at exactly this spot.

The request and indication data blocks as well as the response and confirmation data blocks are identical.

The service description block contains a *result* parameter. If a function returns as positive (result = POS) the service-specific confirmation block will be passed. If the result is negative (result = NEG), then the standard error structure T\_ERROR or a service-specific error structure is passed. Only the initiate service passes a confirmation block even when the result of the function is negative.

If a variable should be transferred to a remote station within a variable length field, the variable has to be given in Motorola format (high order byte first).

For all parameters of data type STRINGV (visible string), byte 0 must contain the length of the character string. (PROFIBUS standard).

As the standard error structure is possible for many services, it is not always noted explicitly. Its structure and the error codes are described in appendix A.

### 3 FMS CONTEXT MANAGEMENT SERVICES

#### 3.1 INITIATE

This service is used to establish a connection between two communication partners: The initiate requester sends context information such as supported services, the supported options, the maximum PDU length and the current version of the OD to its communication partner. Upon receipt of an INITIATE\_REQ\_PDU the FAL of the communication partner checks whether the context of the initiate requester is compatible with its own context.

Various reactions to an INITIATE request are possible. If the communications partner is not available or the FDL Service Access Point are incorrectly parametrized, then FDL answers with an ABORT. If the FAS context check is negative, then LLI responds with an ABORT. Only if the FMS context checks fails or the called application rejects the connection establishment, a negative INITIATE response is returned.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INITIATE
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

*Data Block for Request and Indication:*

Data structure	T_CTXT_INIT_REQ	
USIGN8	profile_number[2]	profile number
PB_BOOL	protection	access protection
USIGN8	password	password
USIGN8	access_groups	access groups
USIGN8	dummy	alignment byte
INT16	od_version	od_version
USIGN8	snd_len_h	max FMS-PDU size to send with high priority
USIGN8	snd_len_l	max FMS-PDU size to send with low priority
USIGN8	rcv_len_h	max FMS-PDU size to receive with high priority
USIGN8	rcv_len_l	max FMS-PDU size to receive with low priority
USIGN8	supported_features[FEAT_SUPP_LEN]	supported features

The user needs to fill in the profile\_number, password and access\_groups parameters. The remaining parameters are prepared by the communication software.

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INITIATE
USIGN8	primitive	RES / CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

Data structure	T_CTXT_INIT_CNF	
USIGN8	profile_number[2]	profile number
INT16	od_version	od version
PB_BOOL	protection	access protection
USIGN8	password	password
USIGN8	access_groups	access groups
USIGN8	dummy	alignment byte

result = NEG:

Data structure	T_CTXT_INIT_ERR_CNF	
USIGN16	class_code	error class and code
USIGN8	snd_len_h	max FMS-PDU size to send with high priority
USIGN8	snd_len_l	max FMS-PDU size to send with low priority
USIGN8	rcv_len_h	max FMS-PDU size to receive with high priority
USIGN8	rcv_len_l	max FMS-PDU size to receive with low priority
USIGN8	supported_features[FEAT_SUPP_LEN]	supported features

The user needs to fill in the profile\_number, password, and access\_groups or class\_code parameters. The remaining parameters are prepared by the communication software.

## 3.2 ABORT

With the ABORT service, the user may release an open connection. The connection may be aborted by the Client or by the Server.

The ABORT service is also called by the protocol stack as reaction to an error situation.

Service-Description-Block for Request and Indication:

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_ABORT
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_CTXT_ABORT_REQ	
PB_BOOL	local	local or remote generated (PB_TRUE = local)
USIGN8	abort_id	abort identifier (USR, LLI_USR, LLI, FDL)
USIGN8	reason	abort reason code
USIGN8	detail_length	length of detail
USIGN8	detail[DETAIL_LENGTH]	detail information about abort reason

Abort Identifiers:

USR	0	identifier USER
FMS	1	identifier FMS (LLI_USR)
LLI	2	identifier LLI
FDL	3	identifier FDL

USER Abort Codes:

USR_ABT_RC1	0	disconnect
USR_ABT_RC2	1	version od incompatible
USR_ABT_RC3	2	password error
USR_ABT_RC4	3	profile number incompatible
USR_ABT_RC5	4	limited service permitted
USR_ABT_RC6	5	od loading interacting permitted

## FMS Abort Codes:

FMS_ABT_RC1	0	FMS-CRL error
FMS_ABT_RC2	1	user error
FMS_ABT_RC3	2	FMS-PDU error
FMS_ABT_RC4	3	connection state conflict LLI
FMS_ABT_RC5	4	LLI error
FMS_ABT_RC6	5	PDU size
FMS_ABT_RC7	6	feature not supported
FMS_ABT_RC8	7	invoke id error response
FMS_ABT_RC9	8	max services overflow
FMS_ABT_RC10	9	connection state conflict FMS
FMS_ABT_RC11	10	service error
FMS_ABT_RC12	11	invoke id error request
FMS_ABT_RC13	12	FMS is disabled

## LLI Abort Codes:

LLI_ABT_RC1	0	LLI context check neg
LLI_ABT_RC2	1	invalid LLI-PDU during associate or abort
LLI_ABT_RC3	2	invalid LLI-PDU during data transfer phase
LLI_ABT_RC4	3	unknown or invalid LLI-PDU received
LLI_ABT_RC5	4	DTA-ACK-PDU received and SAC = 0
LLI_ABT_RC6	5	max no of parallel services exceeded (by LLI)
LLI_ABT_RC7	6	unknown invoke id
LLI_ABT_RC8	7	priority error
LLI_ABT_RC9	8	local error at remote station
LLI_ABT_RC10	9	timeout during associate
LLI_ABT_RC11	10	timeout on cyclic connection
LLI_ABT_RC12	11	timeout of idle receive time
LLI_ABT_RC13	12	error while activating LSAP
LLI_ABT_RC14	13	illegal FDL primitive during ASS or ABT
LLI_ABT_RC15	14	illegal FDL primitive in data transfer
LLI_ABT_RC16	15	unknown FDL primitive
LLI_ABT_RC17	16	unknown LLI primitive
LLI_ABT_RC18	17	illegal LLI primitive during ASS or ABT
LLI_ABT_RC19	18	illegal LLI primitive in data transfer
LLI_ABT_RC20	19	invalid CRL entry
LLI_ABT_RC21	20	ASS connection state conflict
LLI_ABT_RC22	21	procedural error on cyclic connection
LLI_ABT_RC23	22	max no of parallel services exceeded (by FMS)
LLI_ABT_RC24	23	CRL being loaded, LLI is disabled
LLI_ABT_RC25	24	confirm / indication mode error
LLI_ABT_RC26	25	illegal FM1/2 primitive
LLI_ABT_RC27	26	illegal service on cyclic connection
LLI_ABT_RC28	27	FMS-PDU too large on cyclic connection

## LLI Abort Details (AD):

Abort Code	local AD	remote AD
LLI_ABT_RC1		remote LLI context
LLI_ABT_RC2	LLI-PDU type	
LLI_ABT_RC3	LLI_PDU type	
LLI_ABT_RC4	LLI_PDU type	
LLI_ABT_RC10	LLI state	
LLI_ABT_RC18	LLI service	
LLI_ABT_RC19	LLI service	
LLI_ABT_RC27	AD_LLI_INVALID_SERV	
	AD_LLI_INVALID_SERV_CHANGE	
	AD_LLI_INVALID_INDEX_CHANGE	

### FDL Abort Codes:

FDL_ABT_UE	1	remote user interface error
FDL_ABT_RR	2	no remote resources available
FDL_ABT_RS	3	service not activated at remote sap
FDL_ABT_RA	4	no access to remote sap
FDL_ABT_RDL	12	no resource for send response data low
FDL_ABT_RDH	13	no resource for send response data high
FDL_ABT_LS	16	service not activated at local sap
FDL_ABT_NA	17	no reaction from remote station
FDL_ABT_DS	18	disconnected station
FDL_ABT_NO	19	FDL service not OK
FDL_ABT_LR	20	no local resources available
FDL_ABT_IV	21	invalid request parameters

### FDL Abort Details:

FDL_ABT_AD1	0	error while loading update buffer
FDL_ABT_AD2	1	error while activating poll list entry
FDL_ABT_AD3	2	error while deactivating poll list entry
FDL_ABT_AD4	3	transmit error (SDA.con)
FDL_ABT_AD5	4	transmit error (CSR.D.con)
FDL_ABT_AD6	5	transmit error (SRD.con)
FDL_ABT_AD7	6	receive error (CSR.D.con)

### 3.3 REJECT

The REJECT service is used by FMS for rejecting unacceptable PDUs.

#### *Service-Description-Block for Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS_USR
USIGN8	service	FMS_REJECT
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Indication:*

Data structure	T_CTXT_REJECT_IND	
PB_BOOL	detected_here	local or remote detected
INT8	orig_invoke_id	original invoke ID
USIGN8	pdu_type	reject PDU types
USIGN8	reject_code	reject code
PDU types:		
CONFIRMED_REQUEST_PDU	1	confirmed request PDU
CONFIRMED_RESPONSE_PDU	2	confirmed response PDU
UNCONFIRMED_PDU	3	unconfirmed PDU
UNKNOWN_PDU_TYPE	4	unknown PDU type
Reason codes:		
REJ_RC0	0	other
REJ_RC1	1	invoke id exists
REJ_RC2	2	max services overflow
REJ_RC3	3	feature not supported connection oriented
REJ_RC4	4	feature not supported connectionless
REJ_RC5	5	PDU size
REJ_RC6	6	user error connectionless

## 4 VFD SUPPORT SERVICES

### 4.1 LOCAL SERVICES

#### 4.1.1 Create-VFD

This service is used by the user to create a *Virtual Field Device (VFD)* which is assigned to exactly one Object Dictionary. The *vfd\_pointer* parameter in the Communication Relationship List (CRL) entries corresponds exactly to the *vfd\_number* parameter of the CREATE-VFD service.

The following actions have to be done to get a operable FMS:

- one or more VFDs have to be created
- for each VFD an OD has to be loaded
- the CRL has to be loaded

If the CRL header parameter *vfd\_pointer\_supported* is set to PB\_FALSE, then only the first created VFD is valid, and all communication relationships end in this VFD.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_CREATE_VFD_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_VFD_CREATE_REQ	
USIGN32	vfd_number	VFD number
STRINGV	vendor_name[MAX_VFD_STRING_LENGTH]	vendor name
STRINGV	model_name[MAX_VFD_STRING_LENGTH]	model name
STRINGV	revision[MAX_VFD_STRING_LENGTH]	revision of the device
USIGN8	profile_number[2]	profile number



## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	FMS_CREATE_VFD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

Data structure	T_VFD_CREATE_CNF	
USIGN32	vfd_number	VFD number

result = NEG:

Data structure	T_VFD_ERROR	
T_ERROR	error	standard error structure
USIGN32	vfd_number	VFD number

### 4.1.2 VFD-Set-Physical-Status

With this service the user may set the Physical Status of the application in the VFD Object.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_VFD_SET_PHYS_STATUS_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_VFD_SET_PHYS_STATUS_REQ	
USIGN32	vfd_number	VFD number
USIGN8	physical_status	physical status (--> 4.2.2)
USIGN8	dummy	alignment byte

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	FMS_VFD_SET_PHYS_STATUS_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

Data structure	T_VFD_SET_PHYS_STATUS_CNF	
USIGN32	vfd_number	VFD number

result = NEG:

Data structure	T_VFD_ERROR	
T_ERROR	error	standard error structure
USIGN32	vfd_number	VFD number

## 4.2 REMOTE SERVICES

### 4.2.1 Status

The STATUS service is used for reading the status of a remote VFD.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_STATUS
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

n/a

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_STATUS
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

Data structure	T_VFD_STATUS_CNF	
USIGN8	logical_status	logical status (--> 4.2.2)
USIGN8	physical_status	physical status (--> 4.2.2)
USIGN8	local_detail[3]	local detail
USIGN8	dummy	alignment byte

In the Response, the user has to fill in the parameters *local\_detail* and *physical\_status*, the other parameters are set by FMS.

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 4.2.2 Unsolicited-Status

The UNSOLICITED-STATUS service is used by the application (server) to spontaneously transfer a VFD's device/user status. There is no confirmation.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_UN SOLICITED_STATUS
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request and Indication:*

Data structure T\_VFD\_UN SOL\_STATUS\_REQ

USIGN8	priority	LOW / HIGH
USIGN8	logical_status	logical status
USIGN8	physical_status	physical status
USIGN8	dummy1	alignment byte
USIGN8	local_detail[3]	local detail
USIGN8	dummy2	alignment byte

VFD stati:

Logical Status:

STATE_CHANGES_ALLOWED	0	ready to communicate, all services allowed
LIMITED_SERVICES_PERMITTED	2	only limited number of services allowed
OD_LOADING_NON_INTERACTING	4	PUT-OD working, no other PUT-OD allowed at the moment
OD_LOADING_INTERACTING	5	all connections aborted due to PUT-OD

Physical Status:

OPERATIONAL	0	ready
PARTIALLY_OPERATIONAL	1	partially ready
INOPERABLE	2	not ready
NEEDS_COMMISSIONING	3	needs commissioning

### 4.2.3 Identify

The IDENTIFY service selects information for identifying a VFD.

#### *Service-Description-Block for Request and Indication*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_IDENTIFY
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

#### *Data block for Request and Indication:*

n/a

#### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_IDENTIFY
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

#### *Data block for Response and Confirmation:*

result = POS:

Data structure	T_VFD_IDENTIFY_CNF	
STRINGV	vendor_name[MAX_VFD_STRING_LENGTH]	producer of the device
STRINGV	model_name[MAX_VFD_STRING_LENGTH]	model name of the device
STRINGV	revision[MAX_VFD_STRING_LENGTH]	revision of the device

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

If, in the response, the string length (the first byte) is set to zero, the identifier strings are filled in by FMS. Otherwise, the user has to do this job.

## 5 OD MANAGEMENT

### 5.1 LOCAL SERVICES

#### 5.1.1 Loading an Object Dictionary

One or more Object Descriptions are loaded into a local Object Dictionary by means of the service sequence INITIATE-LOAD-OD-LOC, LOAD-OD-LOC and TERMINATE-LOAD-OD-LOC. A distinction is made between loading free of interactions and loading not free of interactions.

The load process is free of interactions if the OD modifications do not affect communication relationships of other stations, e.g. when appending Object Descriptions or deleting private Object Descriptions (*consequence 0*).

The loading process is not free of interactions if other communication relationships access the modified Object Descriptions. In this case the application must abort all communication relationships, then establish them again with the modified OD version number.

If *consequence 1* is specified in the case of loading not free of interactions, then only individual entries can be modified and only the new version number is fetched when the header is loaded. However, for *consequence 2* the whole Object Dictionary is deleted and must be reloaded including the header.

##### 5.1.1.1 Initiate-Load-OD-LOC

This service starts the load process.

*Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_INIT_LOAD_OD_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request:*

Data structure	T_INIT_LOAD_OD_REQ	
USIGN32	vfd_number	VFD number
INT8	consequence	load interacting/ non-interacting
INT8	dummy	alignment byte
Consequence:		
CONSEQUENCE_0	0	non-interacting
CONSEQUENCE_1	1	interacting (only changes)
CONSEQUENCE_2	2	interacting (new OD)

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	FMS_INIT_LOAD_OD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS /NEG

*Data block for Confirmation:*

result = POS:

Data structure	T_INIT_LOAD_OD_CNF	
USIGN32	vfd_number	VFD number

result = NEG:

Data structure	T_SRC_OD_ERROR	
T_ERROR	error	standard error structure
USIGN32	vfd_number	VFD number
USIGN16	index	index

### 5.1.1.2 Load-OD-LOC

This service is used for loading an Object Description in a local Object Dictionary.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_LOAD_OD_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_LOAD_OD_REQ	
USIGN32	vfd_number	VFD number
T_OBJECT_DESCR	obj_descr	object description (--> 5.3.1)

Data structure	T_OBJECT_DESCR	
union		
{		
T_OD_OBJ_DESCR_HDR	od_obj_descr	od object description
T_OD_NULL_OBJECT	null_obj_descr	null object description
T_OD_ST_DT_DESCR	dt_obj_descr	standard data type object description
T_OD_ST_DS_DESCR	ds_obj_descr	standard data structure object description
T_SIMPLE_VAR_OBJECT	s_var_obj_descr	simple variable object description
T_ARRAY_OBJECT	a_var_obj_descr	array variable object description
T_RECORD_OBJECT	r_var_obj_descr	record variable object description
T_VAR_LIST_OBJECT	vlist_obj_descr	variable list object description
T_DOM_OBJECT	dom_obj_descr	domain object description
T_EVENT_OBJECT	evn_obj_descr	event object description
T_PI_OBJ	pi_obj_descr	program invocation object description
} id		

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	FMS_LOAD_OD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG



Data block for Confirmation:

result = POS:

Data structure	T_LOAD_OD_CNF	
USIGN32	vfd_number	VFD number

result = NEG:

Data structure	T_SRC_OD_ERROR	
T_ERROR	error	standard error structure
USIGN32	vfd_number	VFD number
USIGN16	index	index

### 5.1.1.3 Terminate-Load-OD-LOC

This service terminates the load process.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_TERM_LOAD_OD_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_TERM_LOAD_OD_REQ	
USIGN32	vfd_number	VFD number

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	FMS_TERM_LOAD_OD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

Data structure	T_TERM_LOAD_OD_CNF	
USIGN16	vfd_number	VFD number

result = NEG:

Data structure	T_SRC_OD_ERROR	
T_ERROR	error	standard error structure
USIGN32	vfd_number	VFD number
USIGN16	index	index

## 5.1.2 Reading an Object Description

### 5.1.2.1 OD-Read

This service is used for selecting an Object Description from a local Object Dictionary.

*Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_OD_READ_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request:*

Data structure	T_OD_READ_LOC_REQ	
USIGN32	vfd_number	VFD number
USIGN8	obj_code	object code
USIGN8	dummy	alignment byte
T_ACC_SPEC	acc_spec	access specification
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of object
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of object
} id		
ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name
ACCESS_NAME_LIST	2	access by <b>variable list</b> name

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	READ_OD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

Data structure T\_OD\_READ\_LOC\_CNF

USIGN32	vfd_number	VFD number
T_OBJECT_DESCR	obj_descr	object description (--> 5.3.1)

Data structure T\_OBJECT\_DESCR

union		
{		
T_OD_OBJ_DESCR_HDR	od_obj_descr	od object description
T_OD_NULL_OBJECT	null_obj_descr	null object description
T_OD_ST_DT_DESCR	dt_obj_descr	standard data type object description
T_OD_ST_DS_DESCR	ds_obj_descr	standard data structure object description
T_SIMPLE_VAR_OBJECT	s_var_obj_descr	simple variable object description
T_ARRAY_OBJECT	a_var_obj_descr	array variable object description
T_RECORD_OBJECT	r_var_obj_descr	record variable object description
T_VAR_LIST_OBJECT	vlist_obj_descr	variable list object description
T_DOM_OBJECT	dom_obj_descr	domain object description
T_EVENT_OBJECT	evn_obj_descr	event object description
T_PI_OBJ	pi_obj_descr	program invocation object description
} id		

result = NEG:

Data structure T\_SRC\_OD\_ERROR

T_ERROR	error	standard error structure
USIGN32	vfd_number	VFD number
USIGN16	index	index

Since the object codes for all objects are in the third byte, the appropriate data structure of the union can be found by checking this byte.

## 5.2 REMOTE SERVICES

### 5.2.1 Get-OD

One or more Object Descriptions are read with the GET-OD service. The service distinguishes between a short and a long form of Object Descriptions. The long form of the GET-OD service is optional.

Either the name or the index must be specified for reading a single Object Description. Names are searched for only within the object class (see also *access mode*).

For reading more than one, or all Object Descriptions, the index of the first Object Description to be read must be specified (the start index, see also *access mode*). To select the whole Object Dictionary this service must normally be called more than once.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GET_OD
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_GET_OD_REQ	
PB_BOOL	format	PB_TRUE = long, PB_FALSE = short format
USIGN8	dummy	alignment byte
T_ACC_SPEC	acc_spec	access specification
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of object
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of object
} id		

## Access mode:

INDEX_ACCESS	1	access by index
VAR_NAME_ACCESS	2	access by variable name
VAR_LIST_NAME_ACCESS	3	access by variable list name
DOMAIN_NAME_ACCESS	4	access by domain name
PI_NAME_ACCESS	5	access by program invocation name
EVENT_NAME_ACCESS	6	access by event name
START_INDEX_ACCESS	7	access by start index

## Service-Description-Block for Response and Confirmation:

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GET_OD
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

## Data block for Response and Confirmation:

result = POS:<sup>1)</sup>

Data structure	T_GET_OD_CNF:	
PB_BOOL	more_follows	further object descriptions follow
USIGN8	no_of_od_descr	number of object descriptions
T_PACKED_OBJECT_DESCR	packed_object_descr[no_of_od_descr]	packed array of object descriptions (--> 5.3.2)

1) **if the user sets "result = POS" and "no\_of\_od\_descr = 0", the data block is filled in by the communication software!**

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 5.2.2 Put-OD Services

One or more Object Descriptions are written with the PUT-OD services. A distinction is made between loading free of interaction and loading not free of interaction.

The loading process is free of interaction when communication relationships with other stations are not affected by the OD modifications. This is the case for example when appending Object Descriptions or deleting private Object Descriptions. (*consequence 0*).

The loading process is not free of interaction when other communication relationships access the modified Object Descriptions. In this case the application must abort all communication relationships, except for the one from which the PUT-OD service is being executed, then construct them again with the modified OD version number.

If *consequence 1* is specified in the case of loading not free of interaction, then only individual entries can be modified and only the new version number is fetched when the header is loaded. However, for *consequence 2* the whole Object Dictionary is deleted and must be reloaded including the header.

#### 5.2.2.1 Initiate-Put-OD

This service initiates the loading (either with or without retrospective effect) of an Object Dictionary of a remote PROFIBUS station.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INIT_PUT_OD
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

## *Data block for Request and Indication:*

Data structure	T_INIT_PUT_OD_REQ	
INT8	consequence	load interacting / non-interacting
Consequence:		
CONSEQUENCE_0	0	non interacting
CONSEQUENCE_1	1	interacting
CONSEQUENCE_2	2	OD completely new

## *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INIT_PUT_OD
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

## *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------



### 5.2.2.2 Put-OD

The Put-OD service is used for writing one or more Object Descriptions into a remote station's Object Dictionary.

#### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PUT_OD
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

#### *Data block for Request and Indication:*

Data structure	T_PUT_OD_REQ	
USIGN8	dummy	alignment byte
USIGN8	no_of_od_descr	number of object descriptions
T_PACKED_OBJECT_DESCR	packed_object_descr[no_of_od_descr]	packed array of object descriptions (--> 5.3.2)

#### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PUT_OD
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

#### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 5.2.2.3 Terminate-Put-OD

This service terminates the loading of an Object Dictionary into a remote station.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_TERM_PUT_OD
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block:*

n/a

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_TERM_PUT_OD
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 5.3 COMMUNICATION OBJECT STRUCTURE

The following section describes the C-structures of all basic object types.

### 5.3.1 Objects in Local Object Dictionary

A complete Object Description is transferred with the LOAD-OD-LOC or OD-READ service when loading locally or reading the Object Dictionary.

The general Object Description is a union of 11 specific Object Descriptions:

- object dictionary header,
- null object,
- data type,
- structure description of data type,
- simple variable,
- array,
- record,
- variable list,
- domain,
- alarm (event),
- program invocation.

#### General Object Description

Data structure	T_OBJECT_DESCR
union	
{	
T_OD_OBJ_DESCR_HDR	od_obj_descr
T_OD_NULL_OBJECT	null_obj_descr
T_OD_ST_DT_DESCR	dt_obj_descr
T_OD_ST_DS_DESCR	ds_obj_descr
T_SIMPLE_VAR_OBJECT	s_var_obj_descr
T_ARRAY_OBJECT	a_var_obj_descr
T_RECORD_OBJECT	r_var_obj_descr
T_VAR_LIST_OBJECT	vlist_obj_descr
T_DOM_OBJECT	dom_obj_descr
T_EVENT_OBJECT	evn_obj_descr
T_PI_OBJECT	pi_obj_descr
} id	

## Object Dictionary Header

The Object Dictionary Header contains data describing modifiability, symbol length, access rights and version of the Object Dictionary and start indices, length and internal addresses for each of the four parts of the OD:

- Static Type Dictionary
- Static Object Dictionary
- Dynamic Variable List Dictionary
- Dynamic Program Invocation Dictionary

Data structure	T_OD_OBJ_DESCR_HDR	
USIGN16	index	index = 0
USIGN8	obj_code	object-code = 1
PB_BOOL	flag	PB_FALSE = write protected
USIGN8	length	size of names (0-32)
PB_BOOL	protection	access protection supported
INT16	version	version
INT16	len_st_od	length of the static type description
USIGN16	first_index_s_od	start index of the static object description
INT16	len_s_od	length of the static object description
USIGN16	first_index_dv_od	start index of the dyn. variable list description
INT16	len_dv_od	length of the dyn. variable list description
USIGN16	first_index_dp_od	start index of the dyn. pi description
INT16	len_dp_od	length of the dyn. pi description
USIGN32	int_addr	internal address of od description
USIGN32	int_addr_st_od	internal address of the static type description
USIGN32	int_addr_s_od	internal address of the static object description
USIGN32	int_addr_dv_od	internal address of the dyn. variable list description
USIGN32	int_addr_dp_od	internal address of the dyn. pi description

## Null Object

The Null Object is a place holder for standard data types not foreseen in the Static Type Dictionary. Deleted objects are replaced by Null Objects.

Data structure	T_OD_NULL_OBJECT	
USIGN16	index	index
USIGN8	obj_code	object code
USIGN8	dummy	alignment byte

## Data Type Description

Objects of this type represent data type definitions and are stored in the Static Type Dictionary. They cannot be modified in PROFIBUS.

PROFIBUS recognizes 14 predefined data types which are located in indices 1 through 14 in the Static Type Dictionary as follows:

Define	Value	Description
OD_BOOL	1	Boolean
OD_INT8	2	Integer8
OD_INT16	3	Integer16
OD_INT32	4	Integer32
OD_USIGN8	5	Unsigned8
OD_USIGN16	6	Unsigned16
OD_USIGN32	7	Unsigned32
OD_FLOAT	8	Floating Point
OD_VSTRING	9	Visible String
OD_OSTRING	10	Octet String
OD_DATE_TYPE	11	Date Type
OD_TIME_OF_DAY	12	Time of Day
OD_TIME_DIFF	13	Time Difference
OD_BSTRING	14	Bit String

Data structure

T\_OD\_ST\_DT\_DESCR

USIGN16  
USIGN8  
USIGN8  
STRINGV

index  
obj\_code  
dummy  
meaning[MAX\_OBJECT\_NAME\_LENGTH]

index  
object code  
alignment byte  
meaning of the type; for information

## Description of Data Type Structures

Objects of this type describe the structure of compound data types, i.e. records. They are stored in the same place as the Data Type Descriptions, in the Static Type Dictionary. Record objects must contain a reference to such a data type structure description in their Object Description. A single description of a data type structure can be valid for more than one record object.

Data structure	T_OD_ST_DS_DESCR	
USIGN16	index	index
USIGN8	obj_code	object code
USIGN8	no_of_elements	number of record elements
USIGN32	reserved	for internal use
T_OD_DT_LIST	dt_list[no_of_elements]	data type list
Data structure	T_OD_DT_LIST	
USIGN16	index_of_type	index of related type description
USIGN8	length	length of the element in octets
USIGN8	dummy	alignment byte

## Objects in the Static Object Dictionary

Variable Objects (except for Variable Lists), Domain Objects and Event Objects are stored in the Static Object Dictionary.

All Variable services are applicable to the Variable Objects, Event services are applicable to Events and Domain services to Domains.

## Simple Variable

Data structure	T_SIMPLE_VAR_OBJECT	
USIGN16	index	logical address of the object
USIGN8	obj_code	object code
USIGN8	length	length of object in octets
USIGN16	index_of_type	logical address of the type
T_ACCESS	access	access right structure
USIGN32	local_address	local address
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension

## Simple Data Type Field (Array)

Data structure	T_ARRAY_OBJECT	
USIGN16	index	logical address of the object
USIGN8	obj_code	object code
USIGN8	length	length of an element in octets
USIGN16	index_of_type	logical address of the type
USIGN8	nof_elements	number of array elements
USIGN8	dummy	alignment byte
T_ACCESS	access	access right structure
USIGN32	local_address	local address
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension

## Structured Variable (Record)

Data structure	T_RECORD_OBJECT	
USIGN16	index	index
USIGN8	obj_code	object code
USIGN8	no_of_address	number of local addresses
USIGN16	index_of_type	logical address of the type
T_ACCESS	access	access right structure
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension
USIGN32	reserved	for internal use
USIGN32	local_address_list[no_of_address]	local address list

## Variable List

A Variable List is defined as a set of existing objects and represents a new object which is entered in the Dynamic Variable List Dictionary.

Data structure	T_VAR_LIST_OBJECT	
USIGN16	index	logical address of the object
USIGN8	obj_code	object code
USIGN8	no_of_var	number of variables
T_ACCESS	access	access right
PB_BOOL	deletable	PB_TRUE = deletable
USIGN8	dummy	alignment-byte
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension
USIGN32	reserved	for internal use
USIGN16	var_list[no_of_var]	list of variables

## Domain

Data structure	T_DOM_OBJECT	
USIGN16	index	index
USIGN8	obj_code	object code
USIGN8	state	domain state
USIGN8	upload_state	upload state
INT8	counter	in use counter
USIGN16	max_octets	max domain length
T_ACCESS	access	access protection
USIGN32	local_address	local address
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	symbolic name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension

## Event

Data structure	T_EVENT_OBJECT	
USIGN16	index_event	index
USIGN8	obj_code	object code
USIGN8	data_length	size of event data
USIGN16	index_event_data	index of event-data
T_ACCESS	access	access protection
PB_BOOL	enabled	PB_TRUE = event is enabled
USIGN8	dummy	alignment byte
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	symbolic name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension

## Program Invocation

A Program Invocation (PI) is an executable program consisting of one or more Domains which contain program code and data. The first Domain in the associated Domain list must contain executable code. A PI object can be created dynamically and is then entered in the Dynamic Program Invocation Dictionary as a new object.

Data structure	T_PI_OBJECT	
USIGN16	index	pi_index in od
USIGN8	obj_code	object code for OD
USIGN8	cnt_dom	number of domains
T_ACCESS	access	access
PB_BOOL	deletable	deletable
PB_BOOL	reusable	reusable
USIGN8	pi_state	state of pi
USIGN8	dummy	alignment byte
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	symbolic name of pi
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension
USIGN32	reserved	for internal use
USIGN16	dom_list[cnt_dom]	domain index list



## Access Structures for Object Addressing

**Fixed Access Structure (all services except FMS-DEF-VAR-LIST and FMS-PI-CREATE) :**

Data structure	T_ACC_SPEC	
USIGN8	tag	access mode
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of object
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of object
} id		

**Dynamic Access Structure (only for FMS-DEF-VAR-LIST and FMS-PI-CREATE services) :**

Data structure	T_DYN_ACC_SPEC	
USIGN8	tag	access mode
USIGN8	length	length of access specification
USIGN8	acc_spec[length]	access specification (index or name)

**Access mode (all services except FMS-GET-OD, FMS-DEF-VAR-LIST and FMS-PI-CREATE):**

ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name
ACCESS_NAME_LIST	2	access by variable list name

**Access mode for FMS\_DEF-VAR-LIST and FMS-PI-CREATE services:**

DYN_ACCESS_INDEX	3	access by index
DYN_ACCESS_NAME	4	access by symbolic name

**Access mode for FMS-GET-OD service:**

INDEX_ACCESS	1	access by index
VAR_NAME_ACCESS	2	access by variable name
VAR_LIST_NAME_ACCESS	3	access by variable list name
DOMAIN_NAME_ACCESS	4	access by domain name
PI_NAME_ACCESS	5	access by program invocation name
EVENT_NAME_ACCESS	6	access by event name
START_INDEX_ACCESS	7	access by start index

## Access Protection Specification

Data structure                      T\_ACCESS

USIGN8	pass_word	password
USIGN8	acc_groups	access groups
USIGN16	acc_right	access rights

## Object Codes

NULL_OBJECT	0	null object description
OD_OBJECT	1	od object description
DOMAIN_OBJECT	2	domain object description
INVOCATION_OBJECT	3	program invocation object description
EVENT_OBJECT	4	event object description
TYPE_OBJECT	5	data type object description
TYPE_STRUCT_OBJECT	6	data structure object description
SIMPLE_VAR_OBJECT	7	simple variable object description
ARRAY_OBJECT	8	array variable object description
RECORD_OBJECT	9	record variable object description
VAR_LIST_OBJECT	10	variable list object description

Implementation-specific:

VAR_OBJECT	11	Class of all variable objects
ALL_OBJECT	12	Class of all objects

### 5.3.2 PROFIBUS Object Description Transfer

The GET-OD and PUT-OD FMS services can be used to read or modify communication partners' object dictionaries. Both services are able to transfer one or more Object Descriptions. The number of actual Object Descriptions transferred is indicated by the *no\_of\_od\_descr* parameter.

EN 50170/2 (FMS) defines the coding for Object Description services as well as for the basic data types. This coding cannot be directly represented in the local Object Descriptions, since the descriptions are transferred on the bus without gaps and in "Motorola format" (i.e. high byte first).

The GET-OD response is coded by the communication software in the prescribed format in accordance with the data in the Object Dictionary. The GET-OD confirmation contains one or more Object Descriptions in byte string format, to be interpreted by the application according to the PROFIBUS specification.

In the same way the PUT-OD service transfers one or more Object Descriptions as byte strings and the application must handle the structuring and interpretation according to the PROFIBUS specification.

Each individual Object Description consists of a length data item and the actual Object Description itself in accordance with EN 50170/2 (FMS):

Data structure	T_PACKED_OBJECT_DESCR	
USIGN8	length	length of packed object description
USIGN8	packed_obj_descr[length]	packed object description

For correct OD transfer the object descriptions must not contain alignment bytes. Aligning on word delimiters or long word delimiters causes gaps between the length data item and the Object Description, which leads to incorrect PDU formats.

## List of packed Object Descriptions

The mapping of the individual object descriptions onto the parameter packed\_obj\_descr of the services GetOD and PutOD is shown graphically in the following figures. The representation contains two lines. The name of the object attributes are in the upper line. The lower line contains the data types. If the length of the line is insufficient, the representation is continued on the following line.

Using GetOD service in short form the shaded elements in the graphically figures will not be transferred.

### Structure of the OD Object Description

Index	ROM/RAM flag	Name Length	Access Protection Supported	Version OD	Local Address OD-OD	ST-OD Length	Local Address ST-OD	First Index S-OD	S-OD Length
Unsigned16	Boolean	Unsigned8	Boolean	Integer16	Unsigned3 2	Integer16	Unsigned3 2	Unsigned1 6	Integer16

Local Address S-OD	First Index DV-OD	DV-OD Length	Local Address DV-OD	First Index DP-OD	DP-OD Length	Local Address DP-OD
Unsigned32	Unsigned1 6	Integer16	Unsigned32	Unsigned16	Integer16	Unsigned3 2

### Structure of the OD NULL Object Description

Index	Object Code
Unsigned16	Unsigned8

### Structure of the OD Data Type Description

Index	Object Code	Symbolic Name	
		Name Length	Name
Unsigned16	Unsigned8	Unsigned8	Visible String

### Structure of the OD Data Type Structure Description

Index	Object Code	Number of Elements	Data Type Index (1)	Length (1)	..	Data Type Index (n)	Length (n)
Unsigned16	Unsigned8	Unsigned8	Unsigned16	Unsigned8		Unsigned16	Unsigned 8

### Structure of the Simple Variable Object Description

Index	Object Code	Data Type Index	Length	Password	Access Groups	Access Rights	Local Address
Unsigned16	Unsigned8	Unsigned16	Unsigned8	Unsigned8	Bit String 8Bit	Bit String 16Bit	Unsigned32

Variable Name	Extension	
	Length	Data
Visible String	Unsigned8	Octet String

### Structure of the Array Variable Object Description

Index	Object Code	Data Type Index	Length	Number of Elements	Password	Access Groups	Access Rights	Local Address
Unsigned16	Unsigned8	Unsigned16	Unsigned8	Unsigned8	Unsigned8	Bit String 8Bit	Bit String 16Bit	Unsigned32

Array Name	Extension	
	Length	Data
Visible String	Unsigned8	Octet String

**Structure of the Record Variable Object Description**

Index	Object Code	Data Type Index	Password	Access Groups	Access Rights	Record Name	Extension Length      Data	
Unsigned16	Unsigned8	Unsigned16	Unsigned8	Bit String 8Bit	Bit String 16Bit	Visible String	Unsigned8	Octet String

Local Address (1)	..	Local Address (n)
Unsigned32		Unsigned32

**Structure of the Domain Object Description**

Index	Object Code	Max Octets	Password	Access Groups	Access Rights	Local Address	Domain State	Upload State	Counter
Unsigned16	Unsigned8	Unsigned16	Unsigned8	Bit String 8Bit	Bit String 16Bit	Unsigned32	Unsigned8	Unsigned8	Integer8

Domain Name	Extension Length      Data	
Visible String	Unsigned8	Octet String

**Structure of the Event Object Description**

Index	Object Code	Data Type Index	Length	Password	Access Groups	Access Rights	Enabled
Unsigned16	Unsigned8	Unsigned16	Unsigned8	Unsigned8	Bit String 8Bit	Bit String 16Bit	Boolean

Event Name	Extension Length      Data	
Visible String	Unsigned8	Octet String

### Structure of the Variable List Object Description

Index	Object Code	Number of Elements	Password	Access Groups	Access Rights	Deletable	Element Index (1)	..	Element Index (n)
Unsigned16	Unsigned8	Unsigned8	Unsigned8	Bit String 8Bit	Bit String 16Bit	Boolean	Unsigned16		Unsigned16

VarList Name	Extension	
	Length	Data
Visible String	Unsigned8	Octet String

### Structure of the Program Invocation Object Description

Index	Object Code	Number of Domains	Password	Access Groups	Access Rights	Deletable	Reusable	PI State
Unsigned16	Unsigned8	Unsigned8	Unsigned8	Bit String 8Bit	Bit String 16Bit	Boolean	Boolean	Unsigned8

Domain Index (1)	..	Domain Index (n)	PI Name	Extension	
				Length	Data
Unsigned16		Unsigned16	Visible String	Unsigned8	Octet String

## 6 VARIABLE ACCESS

### 6.1 READ

The values of Simple Variables, Arrays and Records of the communication partner may be read by using this service. Single elements of Arrays and Records may be accessed with a subindex.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_READ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure                      T\_VAR\_READ\_REQ

T_ACC_SPEC	acc_spec	access specification
USIGN8	subindex	subindex
USIGN8	dummy	alignment byte

Data structure                      T\_ACC\_SPEC

USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte

union

{		
USIGN16	index	index of object
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of object
} id		

<sup>1)</sup> ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name
ACCESS_NAME_LIST	2	access by variable list name



## Service-Description-Block for Response and Confirmation:

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_READ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

## Data block for Response and Confirmation:

result = POS:

Data structure	T_VAR_READ_CNF	
USIGN8	dummy	alignment byte
USIGN8	length	length of data field in octets
USIGN8	value[length]	data field

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 6.2 WRITE

With this service, values are written in objects of the communication partner. The service may be used for Simple Variables, Arrays, Records and Variable Lists. Single elements of Arrays and Records may be accessed with a subindex.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_WRITE
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_VAR_WRITE_REQ	
T_ACC_SPEC	acc_spec	access specification
USIGN8	subindex	subindex
USIGN8	length	length of data field in octets
USIGN8	value[length]	data field
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of object
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of object
} id		
1) ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name
ACCESS_NAME_LIST	2	access by variable list name

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_WRITE
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 6.3 READ-WITH-TYPE

The values and the Data Type Description of Simple Variables, Arrays and Records of the communication partner may be read by using this service. Single elements of Arrays and Records may be accessed with a subindex.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_READ_WITH_TYPE
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_VAR_READ_WITH_TYPE_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 6.1)
USIGN8	subindex	subindex
USIGN8	dummy	alignment byte

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	READ_WITH_TYPE
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

Data structure	T_VAR_READ_WITH_TYPE_CNF	
USIGN8	no_of_type_descr	number of type descriptions
USIGN8	length	length of data field in octets
T_TYPE_DESCR	type_descr_list[no_of_type_descr]	list of type descriptions
USIGN8	value[length]	data field

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

Data structure	T_TYPE_DESCR	
USIGN8	tag	type description identifier <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
T_SIMPLE_TYPE	simple;	simple type
T_ARRAY_TYPE	array;	array type
T_RECORD_TYPE	record;	record type
} id		
1) SIMPLE_TYPE	1	
ARRAY_TYPE	2	
RECORD_TYPE	3	

Data structure	T_SIMPLE_TYPE	
USIGN16	data_type_index	index of data type
USIGN8	length	size of data type
USIGN8	dummy	alignment byte

Data structure	T_ARRAY_TYPE	
USIGN16	data_type_index	index of data type
USIGN8	length	size of data type
USIGN8	no_of_elements	number of data types

Data structure	T_RECORD_TYPE	
USIGN8	no_of_elements	number of record elements
USIGN8	dummy	alignment byte
T_SIMPLE_TYPE	simple[MAX_VAR_RECORD_ELEMENTS]	list of simple types

## 6.4 WRITE-WITH-TYPE

With this service, values are written in objects of the communication partner. In difference to the WRITE service, a Data Type Description is added to the data to be written. The service may be used for Simple Variables, Arrays, Records and Variable Lists. Single elements of Arrays and Records may be accessed with a subindex.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_WRITE_WITH_TYPE
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Datablock for Request and Indication:*

Data structure	T_VAR_WRITE_WITH_TYPE_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 6.2)
USIGN8	subindex	subindex
USIGN8	dummy	alignment byte
USIGN8	no_of_type_descr	number of type descriptions
USIGN8	length	length of data field in octets
T_TYPE_DESCR	type_descr_list[no_of_type_descr]	type description list (--> 6.3)
USIGN8	value[length]	data field

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_WRITE_WITH_TYPE
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 6.5 INFORMATION-REPORT

With this service, the values of local objects are transmitted to the communication partner. The service may be used for Simple Variables, Arrays, Records and Variable Lists. Single elements of Arrays and Records may be access with the subindex. The execution of this service is not confirmed by the communication partner. Hence, the INFORMATION-REPORT service is functionally equivalent to an unrequested READ response.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INFO_RPT
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_VAR_INFO_RPT_REQ	
USIGN8	priority	LOW / HIGH
USIGN8	subindex	subindex
T_ACC_SPEC	acc_spec	access specification (--> 6.1)
USIGN8	dummy	alignment byte
USIGN8	length	length of data field in octets
USIGN8	value[length]	data field

## 6.6 INFORMATION-REPORT-WITH-TYPE

With this service, the values and the Data Type Descriptions of local objects are transmitted to the communication partner. The service may be used for Simple Variables, Arrays, Records and Variable Lists. Single elements of Arrays and Records may be access with the subindex. The execution of this service is not confirmed by the communication partner. Hence, the INFORMATION-REPORT-WITH-TYPE is functionally equivalent to an unrequested READ-WITH-TYPE response.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INFO_RPT_WITH_TYPE
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_VAR_INFO_RPT_WITH_TYPE_REQ	
USIGN8	priority	LOW / HIGH
USIGN8	subindex	subindex
T_ACC_SPEC	acc_spec	access specification (--> 6.1)
USIGN8	no_of_type_descr	number of type descriptions
USIGN8	length	length of data field in octets
T_TYPE_DESCR	type_descr_list[no_of_type_descr]	type description list (--> 6.3)
USIGN8	value[length]	data field



## 6.7 PHYSICAL-READ

This service is used for reading a communication partner's physical access object's value.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PHYS_READ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_VAR_PHYS_READ_REQ	
USIGN32	Int_addr	physical address to be read
USIGN8	length	length in octets
USIGN8	dummy	alignment byte

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PHYS_READ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

Data structure	T_VAR_PHYS_READ_CNF	
USIGN8	dummy	alignment byte
USIGN8	length	length of data field in octets
USIGN8	data[length]	data field

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 6.8 PHYSICAL-WRITE

This service is used to modify a communication partner's physical access object's value.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PHYS_WRITE
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication*

Data structure	T_VAR_PHYS_WRITE_REQ	
USIGN32	int_addr	physical address to be written to
USIGN8	dummy	alignment byte
USIGN8	length	length of data field in octets
USIGN8	data[length]	data field

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PHYS_WRITE
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 6.9 DEFINE-VARIABLE-LIST

A Variable List Object is created at the communication partner using this service. The client user must ensure that the data of the Variable List service may be transmitted within one PDU.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_DEF_VAR_LIST
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_VAR_DEFINE_VAR_LIST_REQ	
T_ACCESS	access	access rights
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	variable list name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension
USIGN16	index	index of var list (in indication only!)
USIGN8	dummy	alignment byte
USIGN8	no_of_var	number of variables
T_DYN_ACC_SPEC	var_list[no_of_var]	list of variables
Data structure	T_ACCESS	
USIGN8	pass_word	password
USIGN8	acc_groups	access groups
USIGN16	acc_right	access rights
Data structure	T_DYN_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	length	length of access specification
USIGN8	acc_spec[length]	access specification (index or name)
1) DYN_ACCESS_INDEX 3	access by index	
DYN_ACCESS_NAME 4	access by symbolic name	

## *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_DEF_VAR_LIST
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

## *Data block for Response and Confirmation:*

result = POS:

Data structure	T_VAR_DEFINE_VAR_LIST_CNF	
USIGN16	index	index of defined variable list

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 6.10 DELETE-VARIABLE-LIST

This service may be used to delete a Variable List Object at the communication partner.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_DEL_VAR_LIST
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_VAR_DELETE_VAR_LIST_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 6.1)

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_DEL_VAR_LIST
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 6.11 VARIABLE-DATA-EVENT

This service is used to indicate a data event such as a new update of image data memory (IDM) at the Master (cyclic connections).

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS_USR
USIGN8	service	FMS_VAR_DATA_EVENT_LOC
USIGN8	primitive	IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Indication:*

Data structure	T_VAR_DATA_EVENT_IND	
USIGN16	index	variable index
USIGN8	update_ctr	update counter
USIGN8	length	length of data field in octets
USIGN8	value[length]	data field

## 7 DOMAIN-MANAGEMENT SERVICES

With Domain management services it is possible to download Domains from client to server or upload Domains from server to client. The Domain as object is always at the server end. For a given Domain only a download or an upload can be done, but not both at the same time.

### 7.1 DOWNLOAD SERVICES

The download services transfer data Domains from client to server. Note that the initiative for service invocation switches from client to server once the download service has been initialized. The services can only be used for Master-Master communication relationships.

#### 7.1.1 Initiate-Download-Sequence

The INITIATE-DOWNLOAD-SEQUENCE service initializes the download and lets the server know the index or name of the Domain to be loaded.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INIT_DOWNL_SEQ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access specification
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of domain
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of domain
} id		
1)	ACCESS_INDEX	0
	ACCESS_NAME	1
		access by index
		access by symbolic name

*Data block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INIT_DOWNL_SEQ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------



## 7.1.2 Download-Segment

The actual data transport is done with the DOWNLOAD-SEGMENT service. It is initiated by the station which has the Domain as object. The station requests all segments of the Domain in sequence until the *more\_follows* response is set to PB\_FALSE. Segmentation must be done by the application. The data are located in the response-PDU.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_DOWNL_SEG
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.1.1)

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_DOWNL_SEG
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

Data structure	T_DNL_UPL_SEG_CNF	
PB_BOOL	more_follows	PB_TRUE / PB_FALSE (last segment)
USIGN8	data_len	length of data field in octets
USIGN8	data[data_len]	data field

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 7.1.3 Terminate-Download-Sequence

This service terminates a download sequence. It is initiated by the station which has the Domain as object.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_TERM_DOWNL_SEQ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_TERM_DNL_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.1.1)
PB_BOOL	final_result	PB_TRUE / PB_FALSE
USIGN8	dummy	alignment byte

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_TERM_DOWNL_SEQ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 7.1.4 Request-Domain-Download

A server uses the request domain download service to let the client know that a download should be executed. A positive response to this service is sent off only once the entire download sequence has been completed.

#### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_REQ_DOM_DOWNL
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

#### *Data block for Request and Indication:*

Data structure	T_REQUEST_DOM_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.1.1)
USIGN8	dummy	not used
USIGN8	add_info_length	length of additional info in octets
STRINGV	add_info[add_info_length]	additional info

#### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_REQ_DOM_DOWNL
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

#### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 7.2 UPLOAD SERVICES

The upload services are used for transferring data from server to client. In contrast to the download services, each of these services is initiated by the client, that is to say from the station which requests the Domain data but does not have the Domain as object.

### 7.2.1 Initiate-Upload-Sequence

The INITIATE-UPLOAD-SEQUENCE service initiates uploading and lets the server know the index or name of the Domain to be transferred.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INIT_UPL_SEQ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access specification
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of domain
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of domain
} id		
<sup>1)</sup> ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_INIT_UPL_SEQ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 7.2.2 Upload-Segment

The UPLOAD SEGMENT service is used to transfer the data of a Domain from server to client, segment by segment.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_UPL_SEG
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.2.1)

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_UPL_SEG
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

Data structure	T_DNL_UPL_SEG_CNF	
PB_BOOL	more_follows	PB_TRUE / PB_FALSE (last segment)
USIGN8	data_len	length of data field in octets
USIGN8	data[data_len]	data field

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 7.2.3 Terminate-Upload-Sequence

Terminates the uploading process.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_TERM_UPL_SEQ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.2.1)

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_TERM_UPL_SEQ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 7.2.4 Request-Domain-Upload

The server uses the REQUEST-DOMAIN-UPLOAD service to request an upload from a client. A positive response for this service is sent off only once the entire upload sequence has been completed.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_REQ_DOM_UPL
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_REQUEST_DOM_REQ	
T_ACC_SPEC	acc_spec	access spec. (→ 7.2.1)
USIGN8	dummy	alignment byte
USIGN8	add_info_length	length of additional info in octets
STRINGV	add_info[add_info_length]	additional info

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_REQ_DOM_UPL
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------



## 7.3 GENERIC-DOWNLOAD SERVICES

The generic download services are used to load data from the client into the server's Domain. These services can be used for Master-Master and Master-Slave communication relationships. In difference to the standard DOWNLOAD service, the client has always the initiative for the service invocations.

### 7.3.1 Generic-Initiate-Download-Sequence

The GENERIC-INITIATE-DOWNLOAD-SEQUENCE service initializes the download and lets the server know the index or name of the Domain to be loaded.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GEN_INIT_DOWNL_SEQ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access specification
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of domain
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of domain
} id		
1)	ACCESS_INDEX	0
	ACCESS_NAME	1
		access by index
		access by symbolic name

## *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GEN_INIT_DOWNL_SEQ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

## *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 7.3.2 Generic-Download-Segment

The generic download segment service is used to transfer one data segment into the server's domain. The data segment is transmitted in the service request. The client station requests all segments of the domain in sequence and sets the *more\_follows* attribute to PB\_TRUE. In the last request the *more\_follows* attribute has to be set to PB\_FALSE. Segmentation must be done by the application.

#### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GEN_DOWNL_SEG
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

#### *Data block for Request and Indication:*

Data structure	T_GEN_DNL_SEG_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.3.1)
PB_BOOL	more_follows	PB_TRUE / PB_FALSE (last segment)
USIGN8	data_len	length of data field in octets
USIGN8	data[data_len]	data field

#### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GEN_DOWNL_SEG
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

#### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 7.3.3 Generic-Terminate-Download-Sequence

This service terminates a generic download sequence and is initiated by the client station.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GEN_TERM_DOWNL_SEQ
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_DOM_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 7.3.1)

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_GEN_TERM_DOWNL_SEQ
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

Data structure	T_GEN_TERM_DNL_CNF	
PB_BOOL	final_result	final result (PB_TRUE / PB_FALSE))
USIGN8	dummy	alignment byte

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 8 PROGRAM-INVOCATION-MANAGEMENT SERVICES

### 8.1 CREATE-PROGRAM-INVOCATION

This service is used to take Domains which have already been defined in the Object Dictionary and combine them into a program which is then defined in the Dynamic Program Invocation Dictionary as an executable program. It is assumed that the first Domain in the index list contains an executable program.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_CREATE
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_PI_CR8_REQ	
T_ACCESS	access	access rights
USIGN8	cnt_dom	number of domains
PB_BOOL	reusable	PB_TRUE = pi is reusable
USIGN16	index	index of pi (in indication only!)
STRINGV	name[MAX_OBJECT_NAME_LENGTH]	symbolic name
USIGN8	extension[MAX_EXTENSION_LENGTH]	extension
T_DYN_ACC_SPEC	dom_list[cnt_dom]	list of domains
Data structure	T_ACCESS	
USIGN8	pass_word	password
USIGN8	acc_groups	access groups
USIGN16	acc_right	access rights
Data structure	T_DYN_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	length	length of access specification
USIGN8	acc_spec[length]	access specification (index or name)
1) DYN_ACCESS_INDEX 3	access by index	
DYN_ACCESS_NAME 4	access by symbolic name	

## *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_CREATE
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

## *Data block for Response and Confirmation:*

result = POS:

Data structure	T_PI_CR8_CNF	
USIGN16	index	index of pi

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 8.2 DELETE-PROGRAM-INVOCATION

The DELETE-PROGRAM-INVOCATION service deletes a Program Invocation from the Dynamic Program Invocation Dictionary.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_DELETE
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_PI_DEL_REQ	
T_ACC_SPEC	acc_spec	access specification
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of pi
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of pi
} id		
1) ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_DELETE
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure

T\_ERROR

standard error structure



### 8.3 START-PROGRAM-INVOCATION

Used to start up a program.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_START
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_PI_START_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 8.2)
USIGN8	exec_arg[MAX_EXECUTION_ARGUMENT_LENGTH]	execution argument <sup>1)</sup>

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_START
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_PI_ERROR	
T_ERROR	error	standard error structure
USIGN8	pi_state	state of pi
USIGN8	dummy	alignment byte

<sup>1)</sup> The content of the parameter exec\_arg will be transferred as Octet-String. Byte 0 of the parameter exec\_arg contains length of the execution argument.

## 8.4 STOP-PROGRAM-INVOCATION

A running program is halted but not reset to the beginning.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_STOP
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_PI_STOP_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 8.2)

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_STOP
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_PI_ERROR	
T_ERROR	error	standard error structure
USIGN8	pi_state	state of pi
USIGN8	dummy	alignment byte

## 8.5 RESUME-PROGRAM-INVOCATION

A halted program is restarted but not reset to the beginning.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_RESUME
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_PI_RESUME_REQ	
T_ACC_SPEC	acc_spec	access spec. (--> 8.2)
USIGN8	exec_arg[MAX_EXECUTION_ARGUMENT_LENGTH]	execution argument <sup>1)</sup>

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_RESUME
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_PI_ERROR	
T_ERROR	error	standard error structure
USIGN8	pi_state	state of pi
USIGN8	dummy	alignment byte

1) The content of the parameter exec\_arg will be transferred as Octet-String. Byte 0 of the parameter exec\_arg contains length of the execution argument.

## 8.6 RESET-PROGRAM-INVOCATION

A stopped program is reset to its beginning.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_RESET
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_PI_RESET_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 8.2)

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_RESET
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_PI_ERROR	
T_ERROR	error	standard error structure
USIGN8	pi_state	state of pi
USIGN8	dummy	alignment byte

## 8.7 KILL-PROGRAM-INVOCATION

A Program Invocation is aborted, irrespective of its state. The aborted Program Invocation cannot be used again, it can only be deleted.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_KILL
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_PI_KILL_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 8.2)

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_PI_KILL
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 8.8 PI-SET-STATE-LOC

This service is used to adapt the state of the PI-object within the local OD to the state of the real (program) object.

The available states are: UNRUNNABLE, IDLE, RUNNING and STOPPED. The following table shows the states to be accepted depending on the old state. The PI-SET-STATE-LOC service is rejected with a negative confirmation if the state of the PI object is an intermediate state (because a remote service already executed).

Old State	New State	Condition
IDLE	RUNNING UNRUNNABLE	PI locally started PI locally aborted
RUNNING	IDLE STOPPED UNRUNNABLE	"reusable" PI reached end of program PI locally stopped PI locally aborted or reached end of program, but is not "reusable"
STOPPED	RUNNING IDLE UNRUNNABLE	PI locally resumed "reusable" PI locally reset PI locally aborted or locally reset, but is not "reusable"

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS
USIGN8	service	FMS_PI_SET_STATE_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request:*

Data structure	T_PI_SET_STATE_REQ	
USIGN32	vfd_number	vfd number
T_ACC_SPEC	acc_spec	access specification (--> 8.2)
USIGN8	state	new PI state
USIGN8	dummy	alignment byte

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FMS_USR
USIGN8	service	FMS_PI_SET_STATE_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Confirmation:*

result = POS:

Data structure	T_PI_SET_STATE_CNF	
USIGN32	vfd_number	vfd number

result = NEG:

Data structure	T_PI_LOC_ERROR	
T_ERROR	error	standard error structure
USIGN8	pi_state	state of PI
USIGN8	dummy	alignment byte
USIGN32	vfd number	vfd number

## 9 EVENT-MANAGEMENT SERVICES

### 9.1 EVENT-NOTIFICATION

The EVENT-NOTIFICATION service allows an event notification to be sent. This service is not acknowledged by the communication partner.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2.MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_EVN_NOTIFY
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_EVENT_NOTIFY_REQ	
USIGN8	priority	LOW / HIGH
USIGN8	event_number	event count
T_ACC_SPEC	acc_spec	access specification
USIGN8	dummy	alignment byte
USIGN8	data_length	length of data field in octets
USIGN8	event_data[data_length]	data field
Data structure	T_ACC_SPEC	
USIGN8	tag	access mode <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
USIGN16	index	index of event object
STRINGV	name[MAX_ACCESS_NAME_LENGTH]	symbolic name of event object
} id		
1) ACCESS_INDEX	0	access by index
ACCESS_NAME	1	access by symbolic name



## 9.2 EVENT-NOTIFICATION-WITH-TYPE

The EVENT-NOTIFICATION-WITH-TYPE service allows an event notification to be sent. The event notification contains data and their data type description. This service is not acknowledged by the communication partner.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_EVN_NOTIFY_WITH_TYPE
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	POS

Data block for Request and Indication:

Data structure	T_TYPE_DESCR	
USIGN8	tag	type description identifier <sup>1)</sup>
USIGN8	dummy	alignment byte
union		
{		
Data structure	T_EVENT_NOTIFY_WITH_TYPE_REQ	
USIGN8	priority	LOW / HIGH
USIGN8	event_number	event count
T_ACC_SPEC	acc_spec	access specification (--> 9.1)
T_TYPE_DESCR	type_descr	type description
USIGN8	dummy	alignment byte
USIGN8	data_length	length of data field in octets
USIGN8	event_data[data_length]	data field
T_SIMPLE_TYPE	simple;	simple type
T_ARRAY_TYPE	array;	array type
T_RECORD_TYPE	record;	record type
} id		
1) SIMPLE_TYPE	1	
ARRAY_TYPE	2	
RECORD_TYPE	3	

Data structure	T_SIMPLE_TYPE	
USIGN16	data_type_index	index of data type
USIGN8	length	size of data type
USIGN8	dummy	alignment byte
Data structure	T_ARRAY_TYPE	
USIGN16	data_type_index	index of data type
USIGN8	length	size of data type
USIGN8	no_of_elements	number of data types
Data structure	T_RECORD_TYPE	
USIGN8	no_of_elements	number of record elements
USIGN8	dummy	alignment byte
T_SIMPLE_TYPE	simple[MAX_VAR_RECORD_ELEMENTS] list of simple types	

### 9.3 ACKNOWLEDGE-EVENT-NOTIFICATION

This service may be used to acknowledge an Event Notification.

#### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_ACK_EVN_NOTIFY
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

#### *Data block for Request and Indication:*

Data structure	T_ACK_EVN_NOTIFY_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 9.1)
USIGN8	event_number	event count
USIGN8	dummy	alignment byte

#### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_ACK_EVN_NOTIFY
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

#### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 9.4 ALTER-EVENT-CONDITION-MONITORING

The ALTER-EVENT-CONDITION-MONITORING service enables modification (releasing or locking) of event conditions.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_ALT_EVN_CND_MNT
USIGN8	primitive	REQ / IND
INT8	invoke_id	0..127
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_ALT_EVN_CND_MNT_REQ	
T_ACC_SPEC	acc_spec	access specification (--> 9.1)
PB_BOOL	enabled	enable or disable the event
USIGN8	dummy	alignment byte

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	2..MAX_COMREF
USIGN8	layer	FMS / FMS_USR
USIGN8	service	FMS_ALT_EVN_CND_MNT
USIGN8	primitive	RES / CON
INT8	invoke_id	0..127
INT16	result	POS / NEG

### *Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## APPENDIX A

### ERROR STRUCTURE AND ERROR CODES

Data structure:	T_ERROR	
USIGN16	class_code	error class and error code
INT16	add_detail	additional detail
STRINGV	add_description[MAX_ERROR_DESCR_LENGTH]	additional description

The 16 bit *class\_code* parameter contains the error class in the high byte and the error code in the low byte.

	class_ code	error class	error code	meaning class	error
E_FMS_INIT_OTHER	0x0000	0	0	Initiate	Other
E_FMS_INIT_MAX_PDU_SIZE_INSUFF	0x0001	0	1		Max_PDU-Size-insufficient
E_FMS_INIT_FEAT_NOT_SUPPORTED	0x0002	0	2		Feature-Not-Supported
E_FMS_INIT_OD_VERSION_INCOMP	0x0003	0	3		Version-OD-Incompatible
E_FMS_INIT_USER_DENIED	0x0004	0	4		User-Initiate-Denied
E_FMS_INIT_PASSWORD_ERROR	0x0005	0	5		Password-Error
E_FMS_INIT_PROFILE_NUMB_INCOMP	0x0006	0	6		Profile-Number-Incompatible
E_FMS_VFD_STATE_OTHER	0x0100	1	0	VFD-State	Other
E_FMS_APPLICATION_OTHER	0x0200	2	0	Application	Other
E_FMS_APPLICATION_UNREACHABLE	0x0201	2	1		Unreachable
E_FMS_DEF_OTHER	0x0300	3	0	Definition	Other
E_FMS_DEF_OBJ_UNDEF	0x0301	3	1		Object-Undefined
E_FMS_DEF_OBJ_ATTR_INCONSIST	0x0302	3	2		Object-Attributes-Inconsistent
E_FMS_DEF_OBJECT_ALREADY_EXISTS	0x0303	3	3		Object-Already-Exists
E_FMS_RESOURCE_OTHER	0x0400	4	0	Resource	Other
E_FMS_RESOURCE_MEM_UNAVAILABLE	0x0401	4	1		Memory-Unavailable
E_FMS_SERV_OTHER	0x0500	5	0	Service	Other
E_FMS_SERV_OBJ_STATE_CONFLICT	0x0501	5	1		Object-State-Conflict
E_FMS_SERV_PDU_SIZE	0x0502	5	2		PDU-Size
E_FMS_SERV_OBJ_CONSTR_CONFLICT	0x0503	5	3		Object-Constraint-Conflict
E_FMS_SERV_PARAM_INCONSIST	0x0504	5	4		Parameter-Inconsistent
E_FMS_SERV_ILLEGAL_PARAM	0x0505	5	5		Illegal-Parameter
E_FMS_ACCESS_OTHER	0x0600	6	0	Access	Other
E_FMS_ACCESS_OBJ_INVALIDATED	0x0601	6	1		Object-Invalidated
E_FMS_ACCESS_HARDWARE_FAULT	0x0602	6	2		Hardware-Fault
E_FMS_ACCESS_OBJ_ACCESS_DENIED	0x0603	6	3		Object-Access-Denied
E_FMS_ACCESS_ADDR_INVALID	0x0604	6	4		Invalid-Address
E_FMS_ACCESS_OBJ_ATTR_INCONST	0x0605	6	5		Object-Attribute-Inconsistent
E_FMS_ACCESS_OBJ_ACCESS_UNSUPP	0x0606	6	6		Object-Access-Unsupported
E_FMS_ACCESS_OBJ_NON_EXIST	0x0607	6	7		Object-Non-Exist
E_FMS_ACCESS_TYPE_CONFLICT	0x0608	6	8		Type-Conflict
E_FMS_ACCESS_NAME_ACCESS_UNSUP	0x0609	6	9		Name-Access-Unsupported
E_FMS_OD_OTHER	0x0700	7	0	OD	Other
E_FMS_OD_NAME_LEN_ODERFLOW	0x0701	7	1		Name-Length-Overflow
E_FMS_OD_ODERFLOW	0x0702	7	2		OD-Overflow
E_FMS_OD_WRITE_PROTECT	0x0703	7	3		OD-Write-Protected
E_FMS_OD_EXTENSION_LEN_ODERFLOW	0x0704	7	4		Extension-Length-Overflow
E_FMS_OD_OBJ_DESCR_ODERFLOW	0x0705	7	5		OD-Descr-Length-Overflow
E_FMS_OD_OPERAT_PROBLEM	0x0706	7	6		Operational-Problem
E_FMS_OTHER	0x0800	8	0	Other	Other

**INDEX**

Abort Codes, 8  
Abort Identifiers, 8  
Access Mode, 26; 37  
Access Protection, 38  
Array, 35  
Client, 8; 55  
Communication Relationship List, 12  
Data Type Description, 33  
Data Type Structures, 34  
Domain, 36; 55  
Download, 55  
Dynamic Program Invocation Dictionary, 32; 69  
Dynamic Variable List Dictionary, 32  
Event, 36; 80  
Generic Download, 65  
Null Object, 32  
Object Codes, 38  
Object Description, 18; 23; 25  
Object Dictionary, 12; 18; 23; 25  
Object Dictionary Header, 32  
Physical Status, 14  
Predefined Data Types, 33  
Program Invocation, 36; 69  
Record, 35  
Server, 8; 55  
Simple Variable, 34  
Static Object Dictionary, 32  
Static Type Dictionary, 32  
Subindex, 40; 44  
Upload, 60  
Variable List, 35

# **PROFIBUS Application Program Interface**

## **FM7 Services**

Version 5.2  
Rev. 00

Date: 17-October-1997

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---



## CONTENTS

1 SCOPE .....	1
2 OVERVIEW .....	2
3 LOCAL FM7 SERVICES .....	5
3.1 Set-Value-Loc Services.....	5
3.1.1 Set-Busparameter .....	6
3.1.2 Set-Value-Loc .....	8
3.2 Read-Value-Loc Services .....	9
3.2.1 Read-Busparameter .....	9
3.2.2 Read-Value-Loc.....	11
3.3 Loading the Communication Relationship List.....	13
3.3.1 Initiate-Load-CRL-Loc.....	14
3.3.2 Load-CRL-Loc.....	15
3.3.3 Terminate-Load-CRL-Loc .....	17
3.4 Read-CRL-Loc .....	18
3.5 LSAP-Status-Loc.....	20
3.6 Ident-Loc .....	22
3.7 Get-Live-List.....	24
3.8 FM7-Reset.....	26
3.9 FM7-Event.....	27
3.10 FM7-Exit.....	29
4 REMOTE SERVICES .....	30
4.1 FM7-Initiate .....	31
4.2 FM7-Abort .....	33
4.3 Load-CRL-Rem services.....	36
4.3.1 Initiate-Load-CRL-Rem.....	37
4.3.2 Load-CRL-Rem.....	38
4.3.3 Terminate-Load-CRL-Rem .....	40
4.4 Read-CRL-Rem .....	41
4.5 Set-Value-Rem.....	43
4.6 Read-Value-Rem .....	45
4.7 LSAP-Status-Rem.....	47
4.8 Ident-Rem.....	49

5 FM7 CONFIGURATION / COMMUNICATION RELATIONSHIP LIST (CRL).....	51
5.1 CRL Header.....	51
5.2 CRL Entry .....	53
APPENDIX A .....	60
ERROR STRUCTURE AND ERROR CODES .....	60
INDEX .....	61

## 1 SCOPE

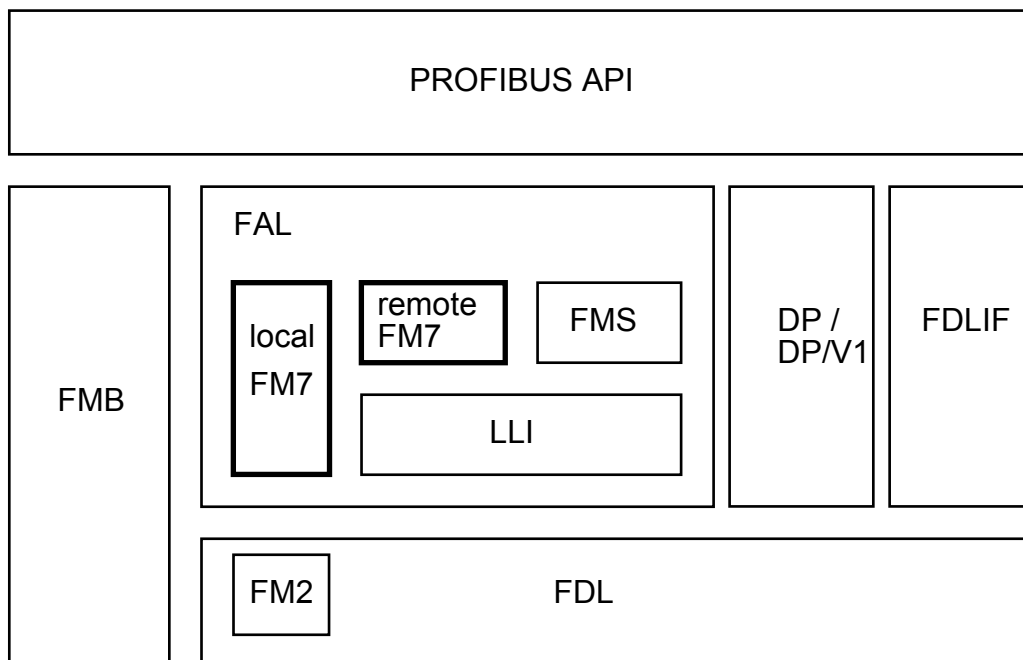
This manual describes the services of local and remote Fieldbus Management Layer 7 (FM7).

The FM7 is part of the Fieldbus Application Layer (FAL). FM7 services provide access to local and remote management objects.

Softing's PROFIBUS Application Layer Interface provides uniform access to all service groups of the PROFIBUS protocol. The common access functions are described in the "User Interface" part of the PROFIBUS User Manual.

This document describes the specific constants, parameters and data structures of all FM7 services.

The FM7-specific types and constants are defined in the include file PB\_FM7.H.



This document should be read in conjunction with the following parts of the PROFIBUS User Manual:

- "User Interface" (describes the uniform access functions to all PROFIBUS services)
- "Basic Management" (describes the management services common to all protocol components)

## **2 OVERVIEW**

The FM7 provides two groups of services. Local FM7 services are used to access local management objects, whereas remote FM7 services carry out management functions over the network and manipulate management objects in remote devices.

Chapter 3 of this document describes the local management services. The remote management services are described in chapter 4. In chapter 5, structure and attributes of the Communication Relationship List (CRL) are described.

General remarks on FM7 services:

- A parallel execution of FM7 services is not possible
- Each device that supports FM7 remote services as server must have a default management connection. This connection is registered in the CRL under communication reference 1. The configuration of the default management connection is specified in EN 50170/2 (FM7).

## Overview of FM7 Services

### Local Management

Service group	Identifier	Code	Page
Set and read FDL bus parameters	FM7_SET_BUSPARAMETER	22	6
	FM7_READ_BUSPARAMETER	24	9
Load and read the CRL	FM7_READ_CRL_LOC	11	18
	FM7_INIT_LOAD_CRL_LOC	12	14
	FM7_LOAD_CRL_LOC	13	15
	FM7_TERM_LOAD_CRL_LOC	14	17
Set and read a single FDL parameter	FM7_SET_VALUE_LOC	15	8
	FM7_READ_VALUE_LOC	16	11
Read status of FDL SAP	FM7_LSAP_STATUS_LOC	17	20
Read identification	FM7_IDENT_LOC	18	22
Read Live-List	FM7_GET_LIVE_LIST	26	24
Event indications from FM2 and LLI	FM7_EVENT	19	27
Reset FAL	FM7_EXIT	21	29
	FM7_RESET	20	26

### Remote Management

Service group	Identifier	Code	Page
FM7 context management	FM7_INITIATE	0	31
	FM7_ABORT	38	33
Load and read the CRL	FM7_READ_CRL_REM	1	41
	FM7_INIT_LOAD_CRL_REM	2	37
	FM7_LOAD_CRL_REM	3	38
	FM7_TERM_LOAD_CRL_REM	4	40
Set and read a single FDL parameter	FM7_SET_VALUE_REM	5	43
	FM7_READ_VALUE_REM	6	45
Read status of FDL SAP	FM7_LSAP_STATUS_REM	7	47
Read identification	FM7_IDENT_REM	8	49

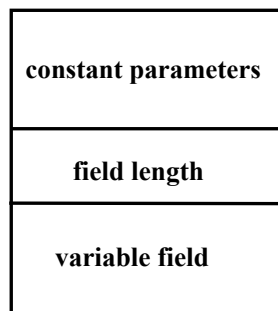
## Notes on Data Structures and Parameters

The FM7-specific types and constants are defined in the include file PB\_FM7.H.

All words, long-words, strings, arrays and records begin on even addresses. To accomplish this, fill bytes had to be added in some places. They are always recognizable by the name *dummy*.

Data blocks do not contain pointers. If a data block contains one or more fields or lists of variable length, then the length information of all variable-length fields is stored in the constant part. The fields of variable length follow on the constant part.

Here is an example of such a data block:



The variable data fields are shown between comment delimiters in the include file PB\_FM7.H to show their position and structure, without forcing the programmer to use data structures of a specific length. Nevertheless, the data must be entered at exactly this spot.

The request and indication data blocks as well as the response and confirmation data blocks are identical.

The service description block contains a *result* parameter. If a function returns as positive (result = POS) the service-specific confirmation block will be passed. If the result is negative (result = NEG), then the standard error structure T\_ERROR or a service-specific error structure is passed. Only the initiate service passes a confirmation block even when the result of the function is negative.

If a variable should be transferred to a remote station within a variable length field, the variable has to be given in Motorola format (high order bytes first).

For all parameters of data type STRINGV (visible string), byte 0 must contain the length of the character string. (PROFIBUS standard).

As the standard error structure is possible for many services, it is not always noted explicitly. Its structure and the error codes are described in appendix A.

### 3 LOCAL FM7 SERVICES

This chapter describes the services of local FM7.

#### 3.1 Set-Value-Loc Services

The PROFIBUS API provides two services to set the FDL operational parameters:

- Set-Busparameters
- Set-Value-Loc

## 3.1.1 Set-Busparameter

This service is used to set all FDL operational parameters that are necessary to start the FDL. This set of operational parameters is called FDL bus parameters.

### Notes:

- In future releases of SOFTING's PROFIBUS API, the service Set-Busparameter will be replaced by service FMB-Set-Busparameter. Set-Busparameter is supported only for compability with former releases of PROFIBUS API. Do not use this service in new applications.
- The FDL bus parameters are described in the FMB manual.

### Service-Description-Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_SET_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### Data block for Request:

Data structure	T_SET_BUSPARAMETER_REQ	
USIGN8	loc_add	local station address
USIGN8	loc_segm	local segment
USIGN8	baud_rate	baud rate
USIGN8	medium_red	medium redundancy
USIGN16	tsl	slot time
USIGN16	min_tsdr	min. station delay time resp.
USIGN16	max_tsdr	max. station delay time resp.
USIGN8	tqui	quiet time
USIGN8	tset	setup time
USIGN32	ttr	target token rotation time
USIGN8	g	gap update factor
PB_BOOL	in_ring_desired	active or passive station
USIGN8	hsa	highest station address
USIGN8	max_retry_limit	max. retry limit
USIGN16	reserved	for internal use
USIGN8	ident[202]	ident

The parameter *ident* is set by the communication software. For compatibility with former releases the component *ident* remains part of the data structure.



### Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_SET_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 3.1.2 Set-Value-Loc

The Set-Value-Loc service is used to set a single FDL operational parameter.

*Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_SET_VALUE_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request:*

Data structure	T_SET_VALUE_REQ	
USIGN8	id	value identifier
USIGN8	length	length of value field
USIGN8	value[length]	value

Parameter identifiers:

FDL-operational parameters which can be changed:

ID_BAUD_RATE	2	Baud rate
ID_TSL	6	Slot-Time
ID_MIN_TSDR	7	Minimum Station Delay Time
ID_MAX_TSDR	8	Maximum Station Delay Time
ID_TQUI	9	Time out
ID_TSET	10	Setup Time
ID_TTR	11	Target Rotation Time
ID_G	12	GAP-Update-Factor
ID_MAX_RETRY_LIMIT	15	Max. # of repetitions in case of transmit error

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_SET_VALUE_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 3.2 Read-Value-Loc Services

The PROFIBUS API offers two services to read the FDL operational parameters:

- Read-Busparameter
- Read-Value-Loc

### 3.2.1 Read-Busparameter

The Read-Busparameter service is used to read the FDL bus parameters.

**Notes:**

- In future releases of SOFTING's PROFIBUS API, the service Read-Busparameter will be replaced by service FMB-Read-Busparameter. Read-Busparameter is supported only for compability with former releases of PROFIBUS API. Do not use this service in new applications.
- The FDL bus parameters are described in the FMB manual.

*Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_READ_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request:*

n/a

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_READ_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

Data structure	T_READ_BUSPARAMETER_CNF	
USIGN8	loc_add	local station address
USIGN8	loc_segm	local segment
USIGN8	baud_rate	baud rate
USIGN8	medium_red	medium redundancy
USIGN16	tsl	slot time
USIGN16	min_tsdr	min. station delay time resp.
USIGN16	max_tsdr	max. station delay time resp.
USIGN8	tqui	quiet time
USIGN8	tset	setup time
USIGN32	ttr	target token rotation time
USIGN8	g	gap update factor
PB_BOOL	in_ring_desired	active or passive station
USIGN8	hsa	highest station address
USIGN8	max_retry_limit	max. retry limit
USIGN16	reserved	not used
USIGN8	ident[202]	FDL ident string

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 3.2.2 Read-Value-Loc

The Read-Value-Loc service allows a single FDL operational parameter to be selected.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_READ_VALUE_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_READ_VALUE_REQ	
USIGN8	id	value identifier
USIGN8	dummy	alignment byte

Parameter identifiers:

FDL operational parameters:

ID_TS	1	Station address
ID_BAUD_RATE	2	Baudrate
ID_MEDIUM_RED	3	Redundancy
ID_HW_RELEASE	4	Hardware release
ID_SW_RELEASE	5	Software release
ID_TSL	6	Slot-Time
ID_MIN_TSDR	7	Minimum Station Delay Time
ID_MAX_TSDR	8	Maximum Station Delay Time
ID_TQUI	9	Time out
ID_TSET	10	Setup Time
ID_TTR	11	Target Rotation Time
ID_G	12	GAP-Update-Factor
ID_IN_RING_DESIRED	13	in ring desired
ID_HSA	14	Highest station address in local segment
ID_MAX_RETRY_LIMIT	15	Max. # of repetitions in case of transmit error
ID_LAS	17	List of active stations (LAS)

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_READ_VALUE_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

Data structure	T_READ_VALUE_CNF	
USIGN8	id	value identifier
USIGN8	length	length of value field
USIGN8	value[length]	value

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

LAS coding:

length	1..126	number of active stations
value[0]	0..126	1st station address
value[1]	0..126	2nd station address
value[n]	0..126	last station address

### **3.3 Loading the Communication Relationship List**

The CRL is loaded by a sequence of services: the sequence starts with the Initiate-Load-CRL-Loc service. Once the Initiate-Load-CRL-Loc service has been executed, the CRL header and subsequently the CRL entries are loaded using the Load-CRL-Loc service. The load sequence is concluded by the Terminate-Load-CRL-Loc service.

If a CRL have been loaded successfully and a reloading is started by the Initiate-Load-CRL-Loc service, all communication relationships except the default management connection are released. The communication relationships stay locked until the Terminate-Load-CRL-Loc service was executed successfully.

The FM7 does not test the CRL for consistency but only for ability to be loaded. Before the CRL is loaded, it should be checked for correctness and consistency by a configuration tool (e.g. SOFTING's FMS configurator).

Structure and attributes of the Communication Relationship List (CRL) are described, in chapter 5.

## 3.3.1 Initiate-Load-CRL-Loc

This service initializes local CRL loading.

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_INIT_LOAD_CRL_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request:*

n/a

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_INIT_LOAD_CRL_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------



### 3.3.2 Load-CRL-Loc

The LOAD-CRL-LOC service loads the CRL header or the static part of a CRL entry. The CRL header is loaded as a CRL entry with communication reference 0.

In order to load the entire CRL, the Load-CRL-Loc service must be called repeatedly. The CRL header is the first entry to be loaded, The CRL entries have to be loaded in ascending sequential order.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_LOAD_CRL_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_LOAD_CRL_REQ	
USIGN16	desired_cr	communication reference to be loaded
union		
{		
T_CRL_HDR	crl_hdr	header of CRL
T_CRL_STATIC	crl_static	static entry of CRL
} id		
Data structure	T_CRL_HDR	
INT16	nr_of_entries	number of CRL entries
USIGN8	poll_sap	local LSAP to be used for poll list
USIGN8	symbol_length	max symbol length
USIGN32	ass_abt_ci	ASS/ABT control interval
PB_BOOL	vfd_pointer_supported	multiple VFDs are supported
USIGN8	dummy	alignment byte

Data structure	T_CRL_STATIC	
USIGN8	loc_lsap	local LSAP
USIGN8	rem_add	remote address
USIGN8	rem_segm	remote segment
USIGN8	rem_lsap	rem. LSAP
USIGN8	conn_type	connection type
USIGN8	lli_sap	LLI sap
USIGN8	multiplier	multiplier on cyclic connections
USIGN8	conn_attr	connection attribute
USIGN8	max_scc	send confirmed counter
USIGN8	max_rcc	receive confirmed counter
USIGN8	max_sac	send acknowledge counter
USIGN8	max_rac	receive acknowledge counter
USIGN32	ci	control interval
USIGN8	max_pdu_snd_high	max FMS/FM7 PDU length send high
USIGN8	max_pdu_snd_low	max FMS/FM7 PDU length send low
USIGN8	max_pdu_rcv_high	max FMS/FM7 PDU length receive high
USIGN8	max_pdu_rcv_low	max FMS/FM7 PDU length receive low
USIGN8	feature_supp[FEAT_SUPP_LEN]	supported features
STRINGV	symbol[MAX_CRL_SYMBOL_LENGTH]	symbolic name of CRL entry
USIGN32	vfd_pointer	VFD number
USIGN8	extension[MAX_CRL_EXTENSION_LENGTH]	CRL extension

Structure and attributes of the Communication Relationship List (CRL) are described, in chapter 5.

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_LOAD_CRL_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 3.3.3 Terminate-Load-CRL-Loc

The TERMINATE-LOAD-CRL-LOC service terminates the load sequence.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_TERM_LOAD_CRL_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

n/a

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_TERM_LOAD_CRL_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_CRL_ERROR	
T_ERROR	error	standard error structure
USIGN16	error_cr	faulty communication reference

### 3.4 Read-CRL-Loc

The Read-CRL-Loc service is used to read the CRL header or a CRL entry.

To read the CRL header, communication reference 0 must be specified.

To read a CRL entry, its communication reference must be specified. FM7 builds the CRL entry from the FM7 and LLI CRL entries in the case of a management connection, and in case of a FMS connection it uses the FMS and LLI CRL entries. If this communication reference is not available, then the response to the service will indicate *result* = *NEG*. To read the entire CRL the READ-CRL-LOC service must be called repeatedly.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_READ_CRL_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_READ_CRL_REQ	
USIGN16	desired_cr	desired communication reference

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_READ_CRL_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### Data block for Confirmation:

result = POS:

Data structure	T_READ_CRL_CNF	
USIGN16	desired_cr	communication reference read
union		
{		
T_CRL_HDR	crl_hdr	CRL header (--> 3.3.2)
T_CRL_ENTRY	crl_entry	CRL entry
} id		
Data structure	T_CRL_ENTRY	
T_CRL_STATIC	crl_static	static part of CRL entry (--> 3.3.2)
T_CRL_DYNAMIC	crl_dynamic	dynamic part of CRL entry
USIGN8	dummy	alignment byte
USIGN8	crl_status_len	length of status field of CRL entry
USIGN8	crl_status[crl_status_len]	status field
Data structure	T_CRL_DYNAMIC	
USIGN8	rem_add	current remote address
USIGN8	rem_segm	current remote segment
USIGN8	rem_lsap	current remote LSAP
USIGN8	scc	send confirmed counter
USIGN8	rcc	received confirmed counter
USIGN8	sac	send acknowledged counter
USIGN8	rac	received acknowledged counter
PB_BOOL	poll_entry_enabled	poll entry flag
USIGN8	master_role	current master role
USIGN8	dummy	alignment byte

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

Structure and attributes of the Communication Relationship List (CRL) are described, in chapter 5.

Status field construction and coding (contains the LLI state machine states) occurs in accordance with the PROFIBUS specification EN 50170/2 (FM7). The construction depends on the type of communication relationship and the number of parallel services. For a detailed description, see EN 50170/2 (FM7).

### 3.5 LSAP-Status-Loc

The LSAP-Status-Loc service is used to request the configuration of a FDL Service Access Point (SAP).

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_LSAP_STATUS_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Request:*

Data structure	T_LSAP_STATUS_REQ	
USIGN8	lsap	local sap
USIGN8	dummy	alignment byte

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_LSAP_STATUS_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data block for Confirmation:*

result = POS:

Data structure	T_LSAP_STATUS_CNF	
USIGN8	access	station address
USIGN8	addr_extension	segment number
USIGN8	sda	SDA service
USIGN8	sdn	SDN service
USIGN8	srd	SRD service
USIGN8	csrd	CSR service

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## FM7 Services

---

Service coding:

Each service field (*sda*, *sdn*, *srd*, *csrd*) contains the arithmetic sum of the parameters "service" and the "role\_in\_service":

service:

SDA_RESERVED	0x00	SDA service
SDN_RESERVED	0x01	SDN service
SRD_RESERVED	0x03	SRD service
CSRD_RESERVED	0x05	CSRD service

role\_in\_service:

INITIATOR	0x00	initiator role
RESPONDER	0x10	responder role
BOTH_ROLES	0x20	initiator / responder
SERVICE_NOT_ACTIVATED	0x30	service not activated

## 3.6 Ident-Loc

The Ident-Loc service allows the FM7 user to request the manufacturer, the software and hardware releases and the characteristics of the PROFIBUS controller. The user may select the identification of FMS, FM7, LLI, FDL or the station characteristics.

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_IDENT_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request:*

Data structure	T_IDENT_REQ	
USIGN8	instance_id	instance identifier
USIGN8	dummy	alignment byte

### Instance Identifier:

ID_FM7	0	FM7
ID_FMS	1	FMS
ID_LLI	2	LLI
ID_FDL	3	FDL
ID_STATION	4	STATION and CHARACTERISTIC

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_IDENT_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG



### Data block for Confirmation:

result = POS:

Data structure	T_IDENT_CNF	
USIGN8	instance_id	instance identifier
USIGN8	dummy	alignment byte
STRINGV	vendor_name[MAX_IDENT_STRING_LENGTH]	vendor name
STRINGV	controller_type[MAX_IDENT_STRING_LENGTH]	controller type
STRINGV	hw_release[MAX_IDENT_STRING_LENGTH]	HW release
STRINGV	sw_release[MAX_IDENT_STRING_LENGTH]	SW release
T_CHARACTERISTICS	characteristics	station characteristic

result = NEG:

T_ERROR	error	standard error structure
---------	-------	--------------------------

Data structure	T_CHARACTERISTICS	
USIGN8	profile_number[2]	profile number
USIGN8	functions_supp[3]	functions supported
USIGN8	dummy	alignment byte
USIGN8	max_pdu_len	max. FMS/FM7 PDU length
USIGN8	dummy	alignment byte
USIGN8	fms_features_supp [6]	FMS features supported
USIGN8	fma7_services_supp [6]	FM7 features supported
USIGN8	max_sap_value	highest LSAP number
USIGN8	max_no_of_saps	max. number of LSAPs
USIGN16	max_comref	max. communication reference
USIGN16	max_crl_len	max. no. of CRL entries
USIGN32	total_len_of_pdu	total length of PDUs
USIGN16	no_of_parallel_serv	max no. of parallel serv.
USIGN16	max_od_index	highest index in OD
USIGN16	max_od_entries	max. no. of OD entries
USIGN8	max_no_vfd	max. no. of VFDs
USIGN8	max_las_entries	max. no. of LAS
USIGN8	min_tsdr	min. station delay time
USIGN8	trdy	ready time
USIGN8	tsdi	station delay time initiator
USIGN8	max_tsdr	station delay responder
USIGN8	tset	setup time
USIGN8	tqui	quiet time
USIGN8	supported_data_types[4]	supported data types
USIGN8	supported_access_rights[3]	supported access rights
USIGN8	supported_var_types	supported variable types
USIGN8	special_functions[2]	special functions
USIGN8	max_od_symbol_length	max length of symbol in OD
USIGN8	max_crl_symbol_length	max length of symbol in CRL

## 3.7 Get-Live-List

This service provides the FM7 user with an up-to-date list of all stations that are functional on the bus. The service is not available for passive stations.

### Note:

- In future releases of SOFTING's PROFIBUS API, the service Get-Live-List will be replaced by service FMB-Get-Live-List. Get-Live-List is supported only for compatibility with former releases of PROFIBUS API. Do not use this service in new applications.

### Service-Description-Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_GET_LIVE_LIST
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### Data block for Request:

n/a

### Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_GET_LIVE_LIST
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### Data block for Confirmation:

result = POS:

Data structure	T_GET_LIVE_LIST_CNF	
USIGN8	dummy	alignment byte
USIGN8	no_of_elements	number of live list elements
T_LIVE_LIST	live_list[no_of_elements]	live list

result = NEG:

T_ERROR	error	standard error structure
---------	-------	--------------------------

## FM7 Services

---

Data structure	T_LIVE_LIST	
USIGN8	station	station address (0..124)
USIGN8	status	current status of station
Stati:		
PASSIVE	0x00	passive station
ACTIVE_NOT_READY	0x01	active station, not ready
ACTIVE_READY	0x02	active station, ready to enter ring
ACTIVE_IN_RING	0x03	active station in ring

## 3.8 FM7-Reset

The FM7-Reset service is used to reset the FAL with its components FM7, FMS, and LLI.

### Note:

- In future releases of SOFTING's PROFIBUS API, the service FM7-Reset will not be supported. FM7-Reset is supported only for compability with former releases of PROFIBUS API. Please, use FM7-Exit instead of FM7-Reset in new applications.

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_RESET
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request:*

n/a

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_RESET
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 3.9 FM7-Event

The FM7-EVENT service indicates errors and events which have been occurred in LLI or FDL.

### *Service-Description-Block for die Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_EVENT
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	POS

### *Data Block:*

Data structure	T_FM7_EVENT_IND	
USIGN16	comm_ref	communication reference
USIGN8	instance_id	instance identifier
USIGN8	reason	event reason
USIGN8	add_detail	additional detail
USIGN8	dummy	alignment byte

Reason:

(see following page)

## Instance identifier:

LLI	2	Lower Layer Interface
FDL	3	Fieldbus Data link Layer

## LLI-Fault-Indications:

LLI_FM7_RC1	1	error during SAP activation
LLI_FM7_RC15	15	SDN failed
LLI_FM7_RC18	18	timeout during associate
LLI_FM7_RC19	19	timeout during abort

## LLI Additional Details:

LLI-Fault-Indication	Additional Detail
LLI_FM7_RC1	FDL status: NO
LLI_FM7_RC15	FDL status: DS
LLI_FM7_RC18	LLI state
LLI_FM7_RC19	LLI state

## FM2 Events:

FM2_FAULT_ADDRESS	1	duplicate address recognized	
FM2_FAULT_PHY	2	physical layer is malfunctioning	(1)
FM2_FAULT_TTO	3	timeout on bus detected	
FM2_FAULT_SYN	4	no receiver synchronization	
FM2_FAULT_OUT_OF_RING	5	local station out of ring	
FM2_GAP_EVENT	6	GAP area has changed	(1)

(1) Not supported by ASPC2

## Additional FM2 Events (Error messages from ASPC2)

FM2_MAC_ERROR	19	fatal MAC error
FM2_HW_ERROR	20	fatal HW error

## 3.10 FM7-Exit

This service terminates the FAL with its components FMS, LLI and FM7. Termination means that all connections are closed and all used resources such as FDL SAPs or send and receive buffers are released.

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7
USIGN8	service	FM7_EXIT
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request:*

n/a

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FM7_USR
USIGN8	service	FM7_EXIT
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Confirmation:*

result = POS:

n/a

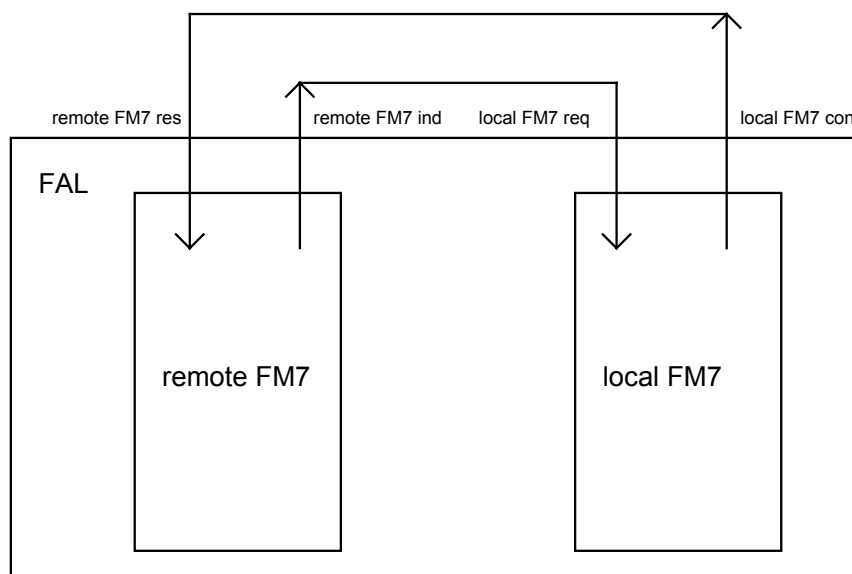
result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 4 REMOTE SERVICES

This chapter describes the service-specific parameters and data for remote FM7 services, i.e. those services which allow communication parameters to be read from or written to remote stations.

All remote FM7 services except for FM7-Initiate and FM7-Abort make use of local management services: after receiving a xxx\_Rem-Indication the FM7 user has to map it onto the corresponding local management service. After receiving the local xxx\_Loc-Confirmation, the data obtained in this way must be transferred to the requesting station with xxx\_Rem-Response.



Of course, the FM7 user may reject a indication or perform additional actions (e.g. store the new values in non-volatile memory).

Each device that supports FM7 remote services as server must have a default management connection. This connection is registered in the CRL under communication reference 1. The configuration of the default management connection is specified in EN 50170/2 (FM7).



## 4.1 FM7-Initiate

FM7 users must establish a management connection using the FM7-Initiate service prior to executing remote management services.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_INITIATE
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_FM7_INIT_REQ	
USIGN8	snd_len_low	max FM7 PDU size to send with low priority
USIGN8	rcv_len_low	max FM7 PDU size to receive with low priority
USIGN8	supported_services [FEAT_SUPP_LEN]	supported FM7 services

The user need not supply any parameters.

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_INITIATE
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Response and Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_FM7_INIT_ERR_CNF	
USIGN16	class_code	error class and code
USIGN8	snd_len_low	max FM7 PDU size to send with low priority
USIGN8	rcv_len_low	max FM7 PDU size to receive with low priority
USIGN8	supported_services [FEAT_SUPP_LEN]	supported FM7 services

If the FM7 user responds negative, he has to fill in the result and the class\_code parameter E\_FM7\_USER\_DENIED (0x0003). The remaining parameters are set by the protocol stack.

## 4.2 FM7-Abort

The FM7 user may release a management connection with the FM7-Abort service.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_ABORT
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_FM7_ABORT_REQ	
PB_BOOL	local	local or remote detected
USIGN8	abort_id	abort identifier USR, LLI_USR (FM7), LLI, FDL
USIGN8	reason	abort reason code
USIGN8	detail_length	length of additional detail
USIGN8	detail[detail_length]	additional detail field

### **Abort Identifiers:**

USR	0	identifier USER
LLI_USR	1	identifier LLI_USR (FM7)
LLI	2	identifier LLI
FDL	3	identifier FDL

### **USER Abort Codes:**

USR_ABT_RC1	0	disconnect
-------------	---	------------

## FM7 Abort Codes:

FM7_ABT_RC1	0	FM7-CRL error
FM7_ABT_RC2	1	user error
FM7_ABT_RC3	2	FM7-PDU error
FM7_ABT_RC4	3	connection state conflict LLI
FM7_ABT_RC5	4	LLI error
FM7_ABT_RC6	5	PDU size
FM7_ABT_RC7	6	feature not supported
FM7_ABT_RC8	7	response error
FM7_ABT_RC9	8	max services overflow
FM7_ABT_RC10	9	connection state conflict FM7
FM7_ABT_RC11	10	service error

## LLI Abort Codes:

LLI_ABT_RC1	0	LLI context check neg
LLI_ABT_RC2	1	invalid LLI-PDU during associate or abort
LLI_ABT_RC3	2	invalid LLI-PDU during data transfer phase
LLI_ABT_RC4	3	unknown or invalid LLI-PDU received
LLI_ABT_RC5	4	DTA-ACK-PDU received and SAC = 0
LLI_ABT_RC6	5	max no of parallel services exceeded (by LLI)
LLI_ABT_RC7	6	unknown invoke id
LLI_ABT_RC8	7	priority error
LLI_ABT_RC9	8	local error at remote station
LLI_ABT_RC10	9	timeout during associate
LLI_ABT_RC11	10	timeout on cyclic connection
LLI_ABT_RC12	11	timeout of idle receive time
LLI_ABT_RC13	12	error while activating LSAP
LLI_ABT_RC14	13	illegal FDL primitive during ASS or ABT
LLI_ABT_RC15	14	illegal FDL primitive in data transfer
LLI_ABT_RC16	15	unknown FDL primitive
LLI_ABT_RC17	16	unknown LLI primitive
LLI_ABT_RC18	17	illegal LLI primitive during ASS or ABT
LLI_ABT_RC19	18	illegal LLI primitive in data transfer
LLI_ABT_RC20	19	invalid CRL entry
LLI_ABT_RC21	20	ASS connection state conflict
LLI_ABT_RC22	21	procedural error on cyclic connection
LLI_ABT_RC23	22	max no of parallel services exceeded (by FMS)
LLI_ABT_RC24	23	CRL being loaded, LLI is disabled
LLI_ABT_RC25	24	confirm / indication mode error
LLI_ABT_RC26	25	illegal FM1/2 primitive
LLI_ABT_RC27	26	illegal service on cyclic connection
LLI_ABT_RC28	27	FMS-PDU too large on cyclic connection

LLI additional abort details if local-flag is FB\_TRUE:

Abort Code	add. detail
LLI_ABT_RC2	LLI-PDU type
LLI_ABT_RC3	LLI_PDU type
LLI_ABT_RC4	LLI_PDU type
LLI_ABT_RC10	LLI state
LLI_ABT_RC18	LLI service
LLI_ABT_RC19	LLI service

LLI additional abort details if local-flag is FB\_FALSE:

Abort Code	add detail
LLI_ABT_RC1	remote LLI context

For all other LLI abort codes there is no additional abort detail.

## FDL Abort Codes:

FDL_ABT_UE	1	remote user interface error
FDL_ABT_RR	2	no remote resources available
FDL_ABT_RS	3	service not activated at remote sap
FDL_ABT_RA	4	no access to remote sap
FDL_ABT_RDL	12	no resource for send response data low
FDL_ABT_RDH	13	no resource for send response data high
FDL_ABT_LS	16	service not activated at local sap
FDL_ABT_NA	17	no reaction from remote station
FDL_ABT_DS	18	local station not in token ring
FDL_ABT_NO	19	FDL service not OK
FDL_ABT_LR	20	no local resources available
FDL_ABT_IV	21	invalid request parameters

## FDL Abort Details:

FDL_ABT_AD1	0	error while loading update buffer
FDL_ABT_AD2	1	error while activating poll list entry
FDL_ABT_AD3	2	error while deactivating poll list entry
FDL_ABT_AD4	3	transmit error (SDA.con)
FDL_ABT_AD5	4	transmit error (CSR.D.con)
FDL_ABT_AD6	5	transmit error (SRD.con)
FDL_ABT_AD7	6	receive error (CSR.D.con)

### **4.3 Load-CRL-Rem services**

In the same way as the local CRL, a CRL is loaded remotely by a sequence of services: the sequence starts with the Initiate-Load-CRL-Rem service. Once the Initiate-Load-CRL-Rem service has been executed, the CRL header and subsequently the CRL entries are loaded using the Load-CRL-Rem service. The load sequence is concluded by the Terminate-Load-CRL-Rem service.

When the Initiate-Load-CRL-Rem service is executed, all server connections except the default management connection are released. The communication relationships stay locked until the Terminate-Load-CRL-Rem service was executed successfully.

The default management connection (communication reference 1) may NOT be transferred when loading the CRL. Nevertheless, it must be taken into account when considering the number of CRL entries in the CRL header. For example, if 10 CRL entries were loaded, then the CRL header must be set to 11 CRL entries, since the default management connection counts as one. The Load-CRL-Rem service is thus invoked 11 times, once with CR=0 for the CRL header and ten times with the communication reference looping from 2 through 11 for the 10 loadable CRL entries.

### 4.3.1 Initiate-Load-CRL-Rem

Initiates loading a CRL onto a remote station.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_INIT_LOAD_CRL_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request and Indication:*

n/a

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_INIT_LOAD_CRL_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Response and Confirmation*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 4.3.2 Load-CRL-Rem

The Load-CRL-Rem service loads the CRL header or static part of a CRL entry onto a remote station. The CRL header is loaded as if it were a CRL entry, with communication reference 0.

The Load-CRL-Rem service must be repeated several times to write the entire CRL. First the CRL header is loaded, then the entries in ascending order beginning with communication reference 2. Communication reference 1 is used for performing the service on the agent (server), hence it must be implicitly available and cannot be loaded by means of the Load-CRL-Rem service.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_LOAD_CRL_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_LOAD_CRL_REQ	
USIGN16	desired_cr	communication reference to be loaded
union		
{		
T_CRL_HDR	cr_hdr	header of CRL(--> 3.3.2)
T_CRL_STATIC	cr_static	static entry of CRL(--> 3.3.2)
} id		



### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_LOAD_CRL_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

### *Data block for Response and Confirmation*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

### 4.3.3 Terminate-Load-CRL-Rem

Terminates loading a CRL onto a remote station.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_TERM_LOAD_CRL_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request and Indication:*

n/a

*Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_TERM_LOAD_CRL_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

*Data block for Response and Confirmation*

result = POS:

n/a

result = NEG:

Data structure	T_CRL_ERROR	
T_ERROR	error	standard error structure
USIGN16	error_cr	faulty communication reference

## 4.4 Read-CRL-Rem

The Read-CRL-Rem service is used for reading the CRL header or a CRL entry from a remote station.

For reading the CRL header, communication reference 0 must be specified. For reading a CRL entry its communication reference must be specified. If the entry is not present, the service confirmation is negative.

For selecting the entire CRL the Read-CRL-Rem service has to be called repeatedly.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_READ_CRL_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_READ_CRL_REQ	
USIGN16	desired_cr	desired communication reference

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_READ_CRL_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## *Data block for Response and Confirmation*

result = POS:

Data structure	T_READ_CRL_CNF	
USIGN16	desired_cr	communication reference read
union		
{		
T_CRL_HDR	crl_hdr	CRL header (--> 3.3.2)
T_CRL_ENTRY	crl_entry	CRL entry (--> 3.4)
} id		

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

The set-up and coding of the status field which contains the LLI state machine condition takes place in accordance with the PROFIBUS EN 50170/2 (FM7, LLI). The field looks different for each communication relationship as its set-up depends on the type of communication relationship and the number of parallel services. No more detailed description is given here, for more information see EN 50170/2 (FM7, LLI).

## 4.5 Set-Value-Rem

The Set-Value-Rem service is used to set FDL operational parameters in remote a station.

*Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_SET_VALUE_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request and Indication:*

Data structure	T_SET_VALUE_REQ	
USIGN8	id	value identifier
USIGN8	length	length of value field
USIGN8	value[length]	value

Parameter identifiers:

FDL operational parameters:

ID_TS	1	Station address	(1)
ID_BAUD_RATE	2	Baud rate	(1)
ID_MEDIUM_RED	3	Redundancy	(1)
ID_HW_RELEASE	4	Hardware release	(1)
ID_SW_RELEASE	5	Software release	(1)
ID_TSL	6	Slot time	
ID_MIN_TSDR	7	Minimum Station Delay Time	
ID_MAX_TSDR	8	Maximum Station Delay Time	
ID_TQUI	9	Timeout	
ID_TSET	10	Setup Time	
ID_TTR	11	Target Rotation Time	
ID_G	12	GAP-Update-Factor	
ID_IN_RING_DESIRED	13	In ring desired	(1)
ID_HSA	14	Highest station address	(1)
ID_MAX_RETRY_LIMIT	15	Max. # of repeats in case of error	

Statistics counters:

ID_FRAME_SENT_COUNT	20	# of messages sent	(1)
ID_RETRY_COUNT	21	# of message repetitions	(1)
ID_SD_COUNT	22	# of valid start delimiters	(1)
ID_SD_ERROR_COUNT	23	# of invalid start delimiter	(1)

Layer 1 parameters

30	Transmitter output (enabled or disabled)	(1)
31	Receiver input (primary or alternate)	(1)
32	Internal send/receive loop (enabled or disabled)	(1)

(1) set value for this parameter is not supported by the management server of Softing's protocol stack version 5.

## *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_SET_VALUE_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## *Data block for Response and Confirmation*

result = POS:

n/a

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## 4.6 Read-Value-Rem

The READ-VALUE-REM service is used to read remote station's FDL operational parameters and statistic counters.

### Service-Description-Block for Request and Indication:

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_READ_VALUE_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### Data block for Request and Indication:

Data structure	T_READ_VALUE_REQ	
USIGN8	id	value identifier
USIGN8	dummy	alignment byte

Parameter identifiers:

FDL operational parameters:

ID_TS	1	Local station address	
ID_BAUD_RATE	2	Baud rate	
ID_MEDIUM_RED	3	Redundancy	
ID_HW_RELEASE	4	Hardware-Release	
ID_SW_RELEASE	5	Software-Release	
ID_TSL	6	Slot-Time	
ID_MIN_TSDR	7	Minimum Station Delay Time	
ID_MAX_TSDR	8	Maximum Station Delay Time	
ID_TQUI	9	Quiet Time	
ID_TSET	10	Setup Time	
ID_TTR	11	Target token Rotation Time	
ID_G	12	GAP-Update-Factor	
ID_IN_RING_DESIRED	13	In ring desired	
ID_HSA	14	Highest station address	
ID_MAX_RETRY_LIMIT	15	Max. # of repetitions in case of error	
ID_TRR	16	Real-Rotation-Time	
ID_LAS	17	List of active Stations (LAS)	
ID_GAPL	18	List of all stations in the local GAP aera	(1)

Statistics counters:

ID_FRAME_SENT_COUNT	20	# of messages sent	(1)
ID_RETRY_COUNT	21	# of repeated messages	(1)
ID_SD_COUNT	22	# of valid start delimiters	(1)
ID_SD_ERROR_COUNT	23	# of incorrect start delimiters	(1)

Layer 1 parameters

30	Transmitter output (enabled or disabled)	(1)
31	Receiver input (primary or alternate)	(1)
32	Internal send/receive loop (enabled or disabled)	(1)

(1) read value for this parameter is not supported by the management server of Softing's protocol stack version 5.

## *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_READ_VALUE_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## *Data block for Response and Confirmation*

result = POS:

Data structure	T_READ_VALUE_CNF	
USIGN8	id	value identifier
USIGN8	length	length of value field
USIGN8	value[length]	value

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

LAS coding:

length	1..126	# of active stations
value[0]	0..126	1st station address
value[1]	0..126	2nd station address
...		
value[n]	0..126	last station address



## 4.7 LSAP-Status-Rem

The LSAP-Status-Rem service is used to request the configuration of a remote station's FDL Service Access Point (SAP).

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_LSAP_STATUS_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_LSAP_STATUS_REQ	
USIGN8	lsap	0..62, BRCT_SAP (63), DEFAULT_SAP (128)
USIGN8	dummy	alignment byte

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_LSAP_STATUS_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Response and Confirmation

result = POS:

Data structure	T_LSAP_STATUS_CNF	
USIGN8	access	station address
USIGN8	addr_extension	segment number
USIGN8	sda	SDA service
USIGN8	sdn	SDN service
USIGN8	srd	SRD service
USIGN8	csrd	CSRD service

result = NEG:

Data structure	T_ERROR	standard error structure
----------------	---------	--------------------------

## Service coding:

Each service field (*sda*, *sdn*, *srd*, *csrd*) contains the arithmetic sum of the parameters "service" and the "role\_in\_service":

service:

SDA_RESERVED	0x00	SDA service
SDN_RESERVED	0x01	SDN service
SRD_RESERVED	0x03	SRD service
CSRD_RESERVED	0x05	CSRD service

role\_in\_service:

INITIATOR	0x00	initiator role
RESPONDER	0x10	responder role
BOTH_ROLES	0x20	initiator / responder
SERVICE_NOT_ACTIVATED	0x30	service not activated

## 4.8 Ident-Rem

The Ident-Rem service is used to get the identification of FMS, FM7, LLI, FDL or the station identification (characteristic) of a remote station.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	2..MAX_COMREF / 1
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_IDENT_REM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	POS

### *Data block for Request and Indication:*

Data structure	T_IDENT_REQ	
USIGN8	instance_id	instance identifier
USIGN8	dummy	alignment byte
Instance identifier:		
ID_FM7	0	FM7
ID_FMS	1	FMS
ID_LLI	2	LLI
ID_FDL	3	FDL
ID_STATION	4	STATION and CHARACTERISTIC

### *Service-Description-Block for Response and Confirmation:*

USIGN16	comm_ref	1 / 2..MAX_COMREF
USIGN8	layer	FM7/FM7_USR
USIGN8	service	FM7_IDENT_REM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data block for Response and Confirmation

result = POS:

Data structure	T_IDENT_CNF	
USIGN8	instance_id	instance identifier
USIGN8	dummy	alignment byte
STRINGV	vendor_name[MAX_IDENT_STRING_LENGTH]	vendor name
STRINGV	controller_type[MAX_IDENT_STRING_LENGTH]	controller type
STRINGV	hw_release[MAX_IDENT_STRING_LENGTH]	HW release
STRINGV	sw_release[MAX_IDENT_STRING_LENGTH]	SW release
T_CHARACTERISTICS	characteristics	station characteristic

result = NEG:

T_ERROR	error	standard error structure
---------	-------	--------------------------

Data structure	T_CHARACTERISTICS	
USIGN8	profile_number[2]	profile number
USIGN8	functions_supp[3]	functions supported
USIGN8	dummy1	alignment byte
USIGN8	max_pdu_len	max. FMS/FM7 PDU length
USIGN8	dummy2	alignment byte
USIGN8	fms_features_supp [6]	FMS features supported
USIGN8	fma7_services_supp [6]	FM7 features supported
USIGN8	max_sap_value	highest LSAP number
USIGN8	max_no_of_saps	max. number of LSAPs
USIGN16	max_comref	max. communication reference
USIGN16	max_crl_len	max. #. of CRL entries
USIGN32	total_len_of_pdu	total length of PDUs
USIGN16	no_of_parallel_serv	max #. of parallel serv.
USIGN16	max_od_index	highest index in OD
USIGN16	max_od_entries	max. #. of OD entries
USIGN8	max_vfd	max. #. of VFDs
USIGN8	max_las_entries	max. #. of LAS
USIGN8	min_tsdr	min. station delay time
USIGN8	trdy	ready time
USIGN8	tsdi	station delay time initiator
USIGN8	max_tsdr	station delay responder
USIGN8	tset	setup time
USIGN8	tqui	quiet time
USIGN8	supported_data_types[4]	supported data types
USIGN8	supported_access_rights[3]	supported access rights
USIGN8	supported_var_types	supported variable types
USIGN8	special_functions[2]	special functions
USIGN8	max_od_symbol_length	max length of symbol in OD
USIGN8	max_crl_symbol_length	max length of symbol in CRL

## 5 FM7 CONFIGURATION / COMMUNICATION RELATIONSHIP LIST (CRL)

The Communication Relationship List (CRL) contains the specific description of all network communication relationships of FAL, independent the time of use. The CRL is structured as a CRL header and CRL entries. The CRL header and each CRL entry are identified by the communication reference (CR).

### 5.1 CRL Header

The CRL header consists of attributes which define the whole CRL. The following figure shows the parameters with their definition, the range of value and the defined constants. The constants are defined in file PB\_FM7.H.

Parameter	Range of Value	Defines	Description
nr_of_entries	0 .. 128	0 .. MAX_COMREF	number of CRL entries
poll_sap	0,2..61 128	DEFAULT_SAP	poll list SAP (see below)
symbol_length	0 .. 32	0 .. CRL_SYMBOL_LENGTH	length of CRL symbol (see below)
ass_abt_ci	1 .. $2^{32}-1$		ASS/ABT control interval (see below)
vfd_pointer_supported	0 255	PB_FALSE PB_TRUE	one VFD supported multiple VFDs supported

Description of the CRL header parameters in detail:

**nr\_of\_entries**

Number of CRL entries in the CRL additional to the CRL header.

**poll\_sap**

Local FDL SAP which contains the FDL Poll List.

**ass\_abt\_ci**

Time control interval in units of 10ms, for monitoring of connection establishment and connection release. The control interval is valid for all CRs.

**symbol\_length**

A symbolic name can be supplied for each entry in the CRL. The symbolic name's maximum length is specified here. The constant CRL\_SYMBOL\_LENGTH is defined in the header file PB\_CONF.H.

**vfd\_pointer\_supported**

This parameter specifies whether one or more VFDs are supported in the CRL.

## 5.2 CRL Entry

The CRL entry contains the complete description of the station's communication relationships. The following figure lists the parameters with their definition, the range of value and the defined constants. The constants are defined in file PB\_FM7.H.

PARAMETER	RANGE OF VALUE	DEFINE	DESCRIPTION
loc_isap	63 128 0 .. 62	BRCT_SAP DEFAULT_SAP	local SAP for receiver of BROADCAST messages for every other case
rem_add	127 255 0 .. 124	ALL GLOBAL_ADDR	remote station address for sender of Broadcast messages used for connections with conn_attr "O_CONN" and broad- or multi-cast CR receivers. for all other cases
rem_segm	0 .. 63 255	NO_SEGMENT	remote segment address Segment addressing not supported
rem_isap	63 255 128 0 .. 62	BRCT_SAP ALL DEFAULT_SAP	remote SAP for sender of BROADCAST messages for connections with "O_CONN" attribute for all other cases for all other cases
conn_type	(see below)	(see below)	connection type
LLI_sap	0 1	FMS_SAP FM7_SAP	LLI SAP for FMS LLI SAP for FM7
multiplier	0 .. 255		poll list multiplier on cyclic connections
conn_attr	0 1 2	D_CONN I_CONN O_CONN	connection attribute defined connection initiator of open connection responder of open connection
max_scc	0 .. 20		max send confirmed counter
max_rcc	0 .. 20		max receive confirmed counter
max_sac	0 .. 20		max send acknowledge counter
max_rac	0 .. 20		max receive acknowledge counter
ci	0 .. $2^{32}-1$		control intervall
max_pdu_snd_high	0 .. 241		max FMS/FM7 PDU length send high
max_pdu_snd_low	0 .. 241		max FMS/FM7 PDU length send low
max_pdu_rcv_high	0 .. 241		max FMS/FM7 PDU length receive low
max_pdu_rcv_low	0 .. 241		max FMS/FM7 PDU length receive low
feature_supp	(see below)		supported FMS/FM7 features
symbol	(see below)		symbolic name
vfd_pointer	0 .. $2^{16}-1$		vfd number
extension	(see below)		extension

Description of the CRL entry parameters in detail:

**loc\_Isap**

In master/slave connections the master is characterized by the fact that its loc\_Isap is the same as the poll list SAP. Hence the slave's loc\_Isap may not be the same as the poll list SAP.

Normally each CR must use a unique loc\_Isap, however, there are two exceptions to this rule:

- 1.) All master/slave CRs of a master use the poll list SAP as their loc\_Isap
- 2.) Several master/master connections for acyclical data traffic (MMAC) can use the same loc\_Isap when only one connection is open at a time. This saves resources. In this case the CR must have the connection attribute "I\_CONN" at the initiating partner's end.

**rem\_add**

This parameter specifies the FDL address of the communication partner. Broadcast- and multicast-receivers may be parameterized for a specific address (0..126) or for all senders (global address ALL). For open connections, the value 255 must be specified for the responder (connection attribute "O\_CONN"). These connections can then be established by any partners.

**rem\_segm**

Segment addressing not supported

**rem\_Isap**

This parameter specifies the FDL Service Access Point of the communication partner.

The rem\_Isap has no meaning for a receiver of a broadcast or multicast communication relationship. Group setup for receivers of a multicast CR takes place by specifying a common loc\_Isap.

Open connections at the responder end (connection attribute "O\_CONN") are also parameterized with rem\_Isap=ALL. These connections can then be established by any loc\_Isap.

In all other cases the communication partner's loc\_Isap must be supplied.



## connection\_type

This parameter contains the components CR-Type, Data-Event and Role-in-Type. The components CR-Type, Data Event and Role in Type are encoded in one byte so that each bit or combination of several bits represents a certain characteristic. The figure below shows the definition of each bit or bit combination.

MSB				LSB			
<b>Data Event</b>		<b>Role in</b>	<b>Type</b>		<b>Cr</b>	<b>Type</b>	

### Cr-Type:

MMAC	0x00	Master/Master acyclic
MSAC	0x01	Master/Slave acyclic
MSAC_SI	0x05	Master/Slave acyclic with slave initiative
MSCY	0x03	Master/Slave cyclic
MSCY_SI	0x07	Master/Slave cyclic with slave initiative
BRCT	0x08	Broadcast
MULT	0x0A	Multicast

### Data-Event:

LLI_N_E	0x00	no Data-Event
LLI_E	0x80	with Data-Event (for master of cyclic master/slave connections)

### Role-in-Type:

MM_RES	0x10	Responder in Master/Master connection
MM_REQ	0x20	Requester in Master/Master connection
MM_REQ_RES	0x30	Requester/Responder in Master/Master connection
MS_RES	0x10	Responder in Master/Slave connection
MS_REQ	0x20	Requester in Master/Slave connection
CL_RCV	0x10	Receiver in Broadcast/Multicast connection
CL_REQ	0x20	Requester in Broadcast/Multicast connection

**max\_scc**

On communication relationships for acyclic data transfer *max\_scc* specifies the maximum number of outstanding confirmations a client may process. On communication relationships for cyclic data transfer and on connectionless communication relationships (BRCT,MULT) *max\_scc* is 0.

Local *max\_scc* must be set in reference with remote *max\_rcc*: (local *max\_scc* ≤ remote *max\_rcc*) must be valid.

**max\_rcc**

On communication relationships for acyclic data transfer *max\_rcc* specifies the maximum number of indications a server may receive without sending a response. On communication relationships for cyclic data transfer and on connectionless communication relationships (BRCT,MULT) *max\_rcc* is 0.

Valid values for *max\_scc* are 0..20; Local *max\_scc* must be set in reference with remote *max\_rcc*: (local *max\_rcc* ≥ remote *max\_scc*) must be valid.

**max\_sac**

This parameter specifies the maximum number of unconfirmed services a sender may send without getting an LLI transport confirmation from the receiver. Local *max\_sac* must be set in reference with remote *max\_rac*: (local *max\_sac* ≤ remote *max\_rac*) must be valid.

**max\_rac**

This parameter specifies the maximum number of unconfirmed services a receiver may receive without sending an LLI transport confirmation. Local *max\_rac* must be set in reference with remote *max\_sac*: (local *max\_rac* ≥ remote *max\_sac*) must be valid.

**CI**

The control interval is interpreted differently for acyclic and cyclic connections.

For acyclic connections, CI indicates the time interval for acyclic connection control (ACI). If there are no messages being transferred on this connection, then an idle message is sent by each of the two stations three times per interval. If either of the two stations does not receive a real or idle message within one interval period, it aborts the connection. The interval must be identical for both stations, and this is checked at connection set-up time.

For cyclic connections, CI indicates the time interval for cyclic connection control. For cyclic connections, besides controlling the connection, the FAL also controls the slave user. To ensure control over the user, cyclic connection control is mandatory. Control is optional for the slave. It only makes sense if a connection breakdown occurs and the slave can execute an action (e.g., failsafe, redundancy switching) in response to the error. The master user's breakdown cannot be recognized by this type of control.

**mult** Poll list multiplier for master in a cyclic master/slave CR

This attribute indicates how often the CR should be entered in the poll list for the master in cyclic data traffic connections. By means of a multiplier > 1 the layer 2 poll interval can be shortened and thus this connection can be given priority over the other cyclic connections. For all other CRs the multiplier has no meaning.

### **max\_pdu\_snd\_high**

This parameter defines the maximum FMS-PDU size for unconfirmed requests with high priority

Note:  $\text{max\_pdu\_snd\_high} \leq \text{partner's max\_pdu\_rcv\_high}$

### **max\_pdu\_snd\_low**

This parameter defines the maximum FMS/FM7-PDU size for requests with low priority

Note:  $\text{max\_pdu\_snd\_low} \leq \text{partner's max\_pdu\_rcv\_low}$

### **max\_pdu\_rcv\_high**

This parameter defines the maximum FMS-PDU size for unconfirmed indications with high priority

Note:  $\text{max\_pdu\_rcv\_high} \geq \text{partner's max\_pdu\_snd\_high}$

### **max\_pdu\_rcv\_low**

This parameter defines the maximum FMS/FM7-PDU size indications with low priority

Note:  $\text{max\_pdu\_rcv\_low} \geq \text{partner's max\_pdu\_snd\_low}$

**Supported FMS / FM7 Features**

The FMS / FM7 features indicates the supported features as client and as server. The server must as a minimum support those services which the client requires.

As these are bit fields, the values are entered in hexadecimal representation. The high value byte is on the right. Bytes which are not used must be assigned with "0x00". The following tables show the values of the possible services.

<b>FMS FEATURES</b>	<b>Byte 0</b>	<b>CLIENT Byte 1</b>	<b>Byte 2</b>	<b>Byte 3</b>	<b>SERVER Byte 4</b>	<b>Byte 5</b>
Get-OD (long form)	0x80			0x80		
Unsolicited-Status	0x40			0x40		
Put-OD	0x20			0x20		
Domain-Download	0x10			0x10		
Generic-Domain-Download	0x10			0x10		
Domain-Upload	0x08			0x08		
Request-Domain-Download	0x04			0x04		
Request-Domain-Upload	0x02			0x02		
Create-Program-Invocation	0x01			0x01		
Delete-Program-Invocation	0x01			0x01		
Start-Program-Invocation		0x80			0x80	
Stop-Program-Invocation		0x80			0x80	
Resume-Program-Invocation		0x80			0x80	
Reset-Program-Invocation		0x80			0x80	
Kill-Program-Invocation		0x40			0x40	
Read		0x20			0x20	
Write		0x10			0x10	
Read-With-Type		0x08			0x08	
Write-With-Type		0x04			0x04	
Physical-Read		0x02			0x02	
Physical-Write		0x01			0x01	
Information-Report			0x80			0x80
Information-Report-With-Type			0x40			0x40
Define-Variable-List			0x20			0x20
Delete-Variable-List			0x20			0x20
Event-Notification			0x10			0x10
Event-Notification-With-Type			0x08			0x08
Acknowledge-Event-Notification			0x04			0x04
Alter-Event-Condition-Monitoring			0x02			0x02
Addressing-By-Name			0x01			0x01

FM7 SERVICES	CLIENT Byte 0	SERVER Byte 1	CLIENT Byte 2	CLIENT Byte 3	SERVER Byte 4	SERVER Byte 5
Initiate-Load-CRL-Rem	0x40	0x40				
Load-CRL-Rem	0x40	0x40				
Terminate-Load-CRL-Rem	0x40	0x40				
Read-CRL-Rem	0x20	0x20				
Set-Value-Rem	0x10	0x10				
Read-Value-Rem	0x08	0x08				
LSAP-Status-Rem	0x04	0x04				
Ident-Rem	0x02	0x02				
reserved	0x00	0x00				

### Symbol

A symbolic name represented through a visible-string (STRINGV) can be entered for each communication relationship. The length of the symbolic name must exceed that specified in the CRL header. If there is no name defined then the symbolic name must be filled with SPACE characters.

### Extension

In addition to the PROFIBUS specification, each communication relationship contains the optional parameter *extension*. The extension can be used to store user specific configuration parameters. The first byte of the extension describes the format of the extension. The following bytes contain the values of user specific configuration parameters.

Encoding of the first extension byte:

MSB

LSB

Extension Bit		Parameter	-Code			Length	Length	Description
0						0	0	no extension available
1						0	0	length in following byte
1						0	1	length 1 Byte
1						1	0	length 2 Bytes
1						1	1	length 3 Bytes

Parameter-Code (5 Bits):

00000:	access protection parameter code
00001:	profile number parameter code
00010 .. 01111:	profile specific parameter code
10000 .. 11111:	vendor specific parameter code

## APPENDIX A

### ERROR STRUCTURE AND ERROR CODES

Data structure: T\_ERROR

USIGN16	class_code	error class and error code
INT16	add_detail	additional detail
STRINGV	add_description[MAX_ERROR_DESCR_LENGTH]	additional description

The 16 bit *class\_code* parameter contains the error class in the high byte and the error code in the low byte.

	class_code	error class	error code	Description Class	Code
E_FM7_APPLICATION_OTHER	0x0100	1	0	Application	Other
E_FM7_APPLICATION_UNREACHABLE	0x0101	1	1		Application-Unreachable
E_FM7_RESOURCE_OTHER	0x0200	2	0	Resource	Other
E_FM7_RESOURCE_MEM_UNAVAILABLE	0x0201	2	1		Memory-Unavailable
E_FM7_SERV_OTHER	0x0300	3	0	Service	Other
E_FM7_SERV_OBJ_STATE_CONFLICT	0x0301	3	1		Object-State-Conflict
E_FM7_SERV_OBJ_CONSTR_CONFLICT	0x0302	3	2		Object-Constraint-Conflict
E_FM7_SERV_PARAM_INCONSIST	0x0303	3	3		Parameter-Inconsistent
E_FM7_SERV_ILLEGAL_PARAM	0x0304	3	4		Illegal-Parameter
E_FM7_SERV_PERM_INTERN_FAULT	0x0305	3	5		Permanent-Internal-Fault
E_FM7_USR_OTHER	0x0400	4	0	User	Other
E_FM7_USR_DONT_WORRY_BE_HAPPY	0x0401	4	1		Don't-Worry-Be-Happy
E_FM7_USR_MEM_UNAVAILABLE	0x0402	4	2		Memory-Unavailable
E_FM7_ACCESS_OTHER	0x0500	5	0	Access	Other
E_FM7_ACCESS_OBJ_ACC_UNSUP	0x0501	5	1		Object-Access-Unsupported
E_FM7_ACCESS_OBJ_NON_EXIST	0x0502	5	2		Object-Non_Existent
E_FM7_ACCESS_OBJ_ACCESS_DENIED	0x0503	5	3		Object-Access-Denied
E_FM7_ACCESS_HARDWARE_FAULT	0x0504	5	4		Hardware-Fault
E_FM7_ACCESS_TYPE_CONFLICT	0x0505	5	5		Type-Conflict
E_FM7_CRL_OTHER	0x0600	6	0	CRL	Other (see additional detail)
E_FM7_CRL_INVALID_ENTRY	0x0601	6	1		Invalid-CRL-Entry
E_FM7_CRL_NO_CRL_ENTRY	0x0602	6	2		No-CRL-Entry
E_FM7_CRL_INVALID_CRL	0x0603	6	3		Invalid-CRL
E_FM7_CRL_NO_CRL	0x0604	6	4		No-CRL
E_FM7_CRL_WRITE_PROTECTED	0x0605	6	5		CRL-Write-Protected
E_FM7_CRL_NO_ENTRY_FOUND	0x0606	6	6		No-CRL-Entry-Found
E_FM7_CRL_NO_MULT_VFD_SUPP	0x0607	6	7		Multiple-VFDs-Not-Supported
E_FM7_OTHER	0x0700	7	0	Other	Other

#### Additional-Details:

NO_ADD_DETAIL	0x00
AD_LLI_LSAP_ACT_FAILED	0x02

## **INDEX**

Abort Codes, 33  
Abort Identifiers, 33  
Bus Parameters, 6  
Communication Relationship List, 51  
CRL Entry, 53  
CRL Header, 51  
Default Management Connection, 36  
Error Codes, 60  
Error Structure, 60  
Local Management, 5  
Management Connection, 31  
Management Events, 27  
Remote Management, 30





# **PROFIBUS Application Program Interface**

## **DP Services**

Version 5.2  
Rev. 02

Date: 29-June-2001

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

---

## Copyright Notice

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

## CONTENTS

1 SCOPE .....	1
2 OVERVIEW .....	2
2.1 FEATURES .....	2
2.2 CONCEPT .....	3
3 FUNCTIONALITY .....	6
3.1 ARCHITECTURE .....	6
3.2 BASIC STATE MACHINE .....	7
3.3 SOFTWARE CONFIGURATION .....	8
3.4 DP MASTER PARAMETER SET .....	9
3.5 DP SLAVE PARAMETER SETS .....	11
3.6 DPRAM ADDRESS ASSIGNMENT MODES .....	12
3.6.1 "ARRAY" Mode .....	13
3.6.2 "USER DEFINED" Mode .....	14
3.6.3 "COMPACT" Mode .....	15
3.6.4 "IO-BLOCK" Mode .....	16
3.7 LOCAL / REMOTE SERVICES .....	18
4 SERVICE INTERFACE .....	20
4.1 OVERVIEW .....	20
4.2 INITIALIZATION / TERMINATION .....	23
4.2.1 Init_Master .....	23
4.2.2 Exit_Master .....	25
4.3 DP MASTER (CLASS 1) SERVICE INTERFACE .....	26
4.3.1 Upload_Loc / Download_Loc .....	27
4.3.2 Start_Seq_Loc / End_Seq_Loc .....	30
4.3.3 Act_Param_Loc .....	32
4.3.4 Data_Transfer .....	34
4.3.5 Get_Slave_Diag .....	36
4.3.6 Set_Prm_Loc .....	39
4.3.7 Get_Master_Diag_Loc .....	40
4.3.8 Get_Slave_Param .....	42
4.3.7 Set_Busparameter .....	44
4.4 DP MASTER (CLASS 2) SERVICE INTERFACE .....	45
4.4.1 Upload / Download .....	45
4.4.2 Start_Seq / End_Seq .....	48
4.4.3 Act_Para_Brct .....	51
4.4.4 Act_Param .....	52
4.4.5 Get_Master_Diag .....	54

4.5 DDLM SERVICE INTERFACE .....	55
4.5.1 Set_Prm .....	56
4.5.2 Chk_Cfg .....	57
4.5.3 Get_Cfg .....	58
4.5.4 Slave_Diag .....	59
4.5.5 RD_Inp / RD_Outp .....	61
4.5.6 Data_Exchange .....	62
4.5.7 Global_Control .....	63
4.5.8 Set_Slave_Add .....	64
5 DATA INTERFACE .....	65
5.1 DP SLAVE I/O DATA ACCESS .....	67
5.2 STATUS INFORMATION .....	67
6. DP STATUS AND ERROR CODES .....	69
6.1. CODING CONVENTIONS .....	69
6.2. ERROR CODE DEFINITIONS .....	69
6.2.1. Error Codes .....	69
6.2.2. Error Code Extensions .....	70

## 1 SCOPE

This document describes the concept of Softing's PROFIBUS DP protocol implementation and specifies the programming interface between PROFIBUS DP protocol software and PROFIBUS DP application.

This document does not describe the functionality of PROFIBUS DP. Therefore, it is expected that the reader is familiar with PROFIBUS DP and that he knows EN 50170/2.

Softing's PROFIBUS Application Program Interface provides uniform access to all service groups of the PROFIBUS protocol. The common access functions are described in the "User Interface" part of the PROFIBUS User Manual.

This document describes the specific constants and data structures of all DP services.

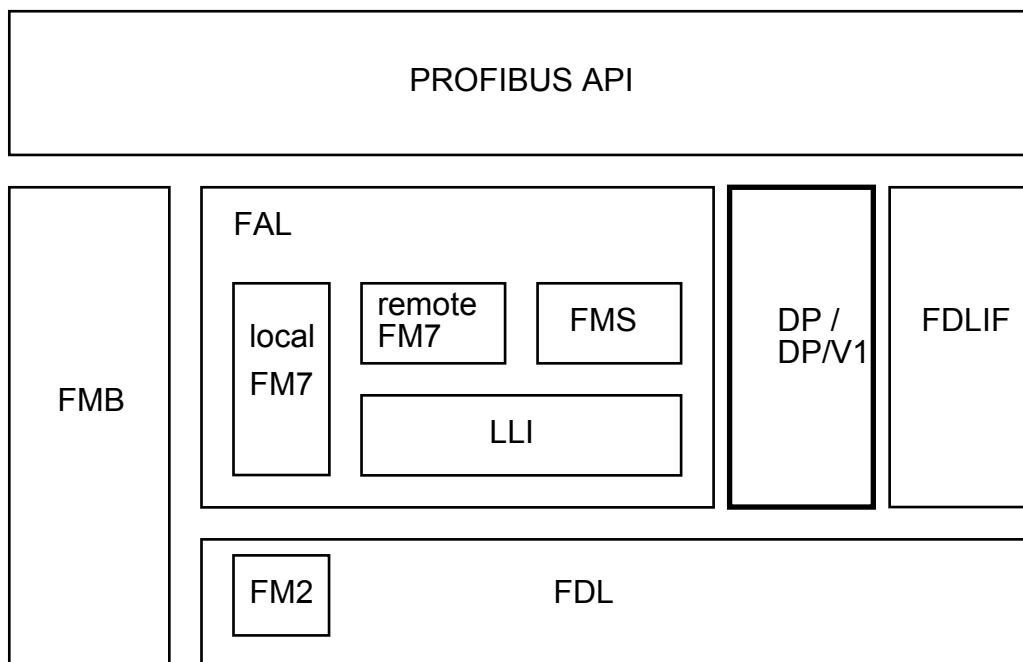


Figure 1-1: Softing PROFIBUS protocol stack architecture

This document should be read in conjunction with the following parts of the PROFIBUS User Manual:

- "User Interface" (describes the uniform access functions to all PROFIBUS services)
- "FMB Services" (describes the management services which are necessary to configure the PROFIBUS application layer)

## **2 OVERVIEW**

### **2.1 FEATURES**

Softing's PROFIBUS DP protocol software acts as an active node within a PROFIBUS fieldbus network with distributed I/O devices (decentral periphery). It might be used as a DP Slave polling device "DP Master (class 1)" or as a configuration and parameterization tool "DP Master (class 2)" or both simultaneously.

The rigid conformity to the standard EN 50170/2 guarantees interoperability in multi-vendor PROFIBUS networks. Softing's DP protocol software supports the entire scope of services defined in the standard.

The software is adaptable to different requirements of user applications. That implies optimal use of the available resources and a maximum of performance.

All features of PROFIBUS regarding bus transmission speed, telegram length or various station addresses are supported.

Softing's PROFIBUS Application Program Interface (PAPI) is the comfortable way to send and receive network messages or to use the DP Slave process and diagnostic data.

The PROFIBUS DP protocol software offers the following features:

- complete DP Master (class 1) functionality including responder functionality to DP Masters (class 2)
- complete function set of DP Masters (class 2) acting as requester (configuration device)
- no limitations regarding station address, telegram length or diagnostic data image of any DP Slave (depending on the available memory amount)
- support of I/O data image in Dual Ported Memory (DPRAM) via Softing's PROFIBUS Application Program Interface
- fragmentation and location of the DP Slave I/O data image is definable by the user application (Address Assignment Modes)
- all PROFIBUS DP Master/Slave services are implemented (i.e. DP Slave station address assignment, acyclic read of inputs and outputs)
- additional status information about DP Slave errors and diagnostic data during every polling cycle is available and analysed by the Master

## 2.2 CONCEPT

### Combi Stack Architecture

The implementation comprises the functionality of PROFIBUS DP Master devices class 1 and class 2 in accordance with EN 50170/2.

The protocol stack is designed to allow mixed operation of PROFIBUS DP with all other PROFIBUS components (FAL, FDLIF, SM7). Distinction between different services is made only by use of different Service Access Points (SAPs). Stand-alone PROFIBUS DP versions are supported for more efficient and faster applications. The Basic Management (FMB) is responsible for controlling shared FDL access and the allocation of resources.

### PROFIBUS Application Program Interface, Dual Ported Memory

In order to achieve uniform access to different protocol stacks the PROFIBUS Application Program Interface is used also by DP. Thus, Service Description Blocks (SDBs) are set up to describe the service independent parts of request commands, and Data Blocks (DBs) to specify the service specific parts. To address the new PROFIBUS DP services, a new layer identifier must be used within the SDBs. The document "PROFIBUS Application Program Interface (PAPI) - User Interface" gives a detailed explanation of this mechanism.

The PROFIBUS DP service interface is accessible either via Dual Ported Memory (DPRAM) by means of the PAPI functionality or via a task interface without DPRAM. The content of the service description and data blocks is always identical (described in this document).

The PROFIBUS Application Program Interface provides a Data Interface that is used to access the input and output area of each DP Slave. Optionally status information for each station is supported. This information is updated cyclically and informs about the correct data transfer or available diagnostic data during the last polling cycle. The user application gets the opportunity to determine which DP Slave output areas should be transferred or not.

To synchronise the data transfer phases of the DP Master (class 1) and the user application, a specific service is provided. In this implementation the user application is responsible for issuing "Data\_Transfer" requests to start a new polling cycle. The related confirmation indicates the end of the polling cycle.

### Service Mapping

All services of DP Master (class 2) are defined in the same way as specified in EN 50170/2. To ease the use of local and remote services in terms of identical functionality all remote services have a local counterpart. That means the user application may pass the incoming remote service directly to the local entity by means of the respective local service (i.e. "Download - Download\_Loc", "Act\_Param - Act\_Param\_Loc", etc.). This approach guarantees convenient access to the communication services nearly independent from the user's role (client / server). Also the user is able to use the remote functionality locally without having different service structures (i.e. getting master status information).

## Diagnostic Data

When considering the case of different processing times in the communication stack and in the user application, diagnostic data must not be lost at the interface between them. That is the reason why a buffer mechanism is provided to supply the user application with diagnostic data. DP Slave diagnostic data is entered in a circular buffer. An adjustable number of diagnostic data messages can be held as history. The oldest diagnostic entry is overwritten and an overrun flag is set when a buffer overflow occurs.

To get a diagnostic data snapshot from all DP Slaves currently being used the user must create its own diagnostic data image. Once initialised at startup only the newly arriving diagnostic messages must be updated in that chart.

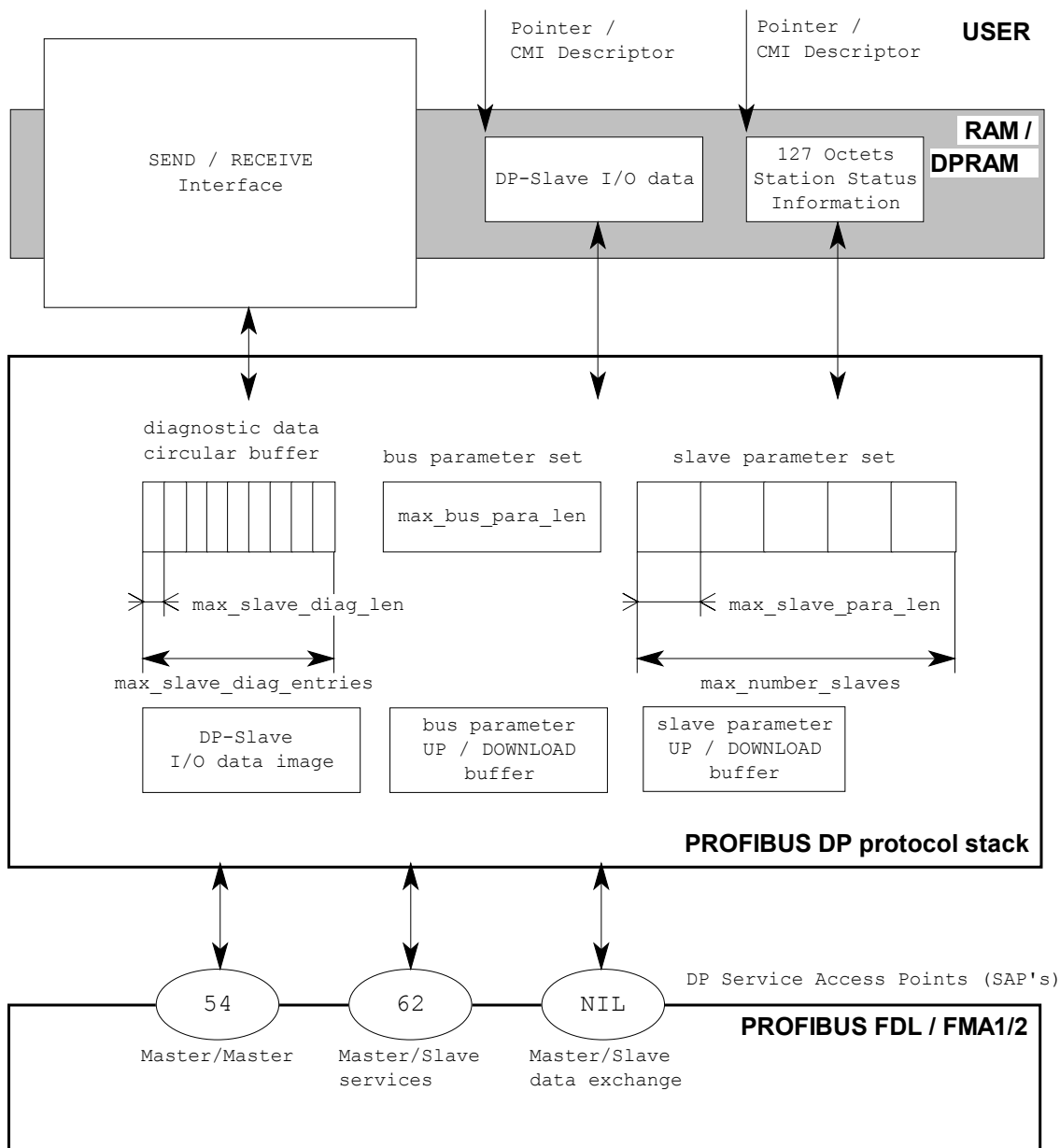


Figure 2-1: overview PROFIBUS DP protocol stack



### DP Slave I/O data layout

Usually the I/O data channels of each DP Slave are located equidistant in the (Dual Ported) RAM area. The DP Slave data then will be handled as data array (one I/O channel per station). DP Slaves using only a part of the allocated I/O memory consequently leave gaps within the DPRAM.

Since user applications depend on the requirements of the automation task, other layouts of I/O areas might be necessary. Therefore two other memory Address Assignment Modes (AAM) are supported. By means of an Address Assignment Table (AAT - part of a Slave Parameter Set) the location and fragmentation of the DP Slave I/O areas is user definable. Thus also the amount of memory used for the communication participants is changeable.

## 3 FUNCTIONALITY

### 3.1 ARCHITECTURE

The PROFIBUS DP protocol stack consists of two main parts:

- **User Interface (USIF):** built-in application that takes care of each activated DP Slave; parameterizes and configures the DP Slaves and supplies automatically I/O and diagnostic data
- **Direct Data Link Mapper (DDLM):** sublayer for convenient FDL access; offers nearly all DP services and handles Master/Master communication

There was not made any distinction between services of different sublayers. The user may always direct his service requests to the "DP" layer (not USIF or DDLM).

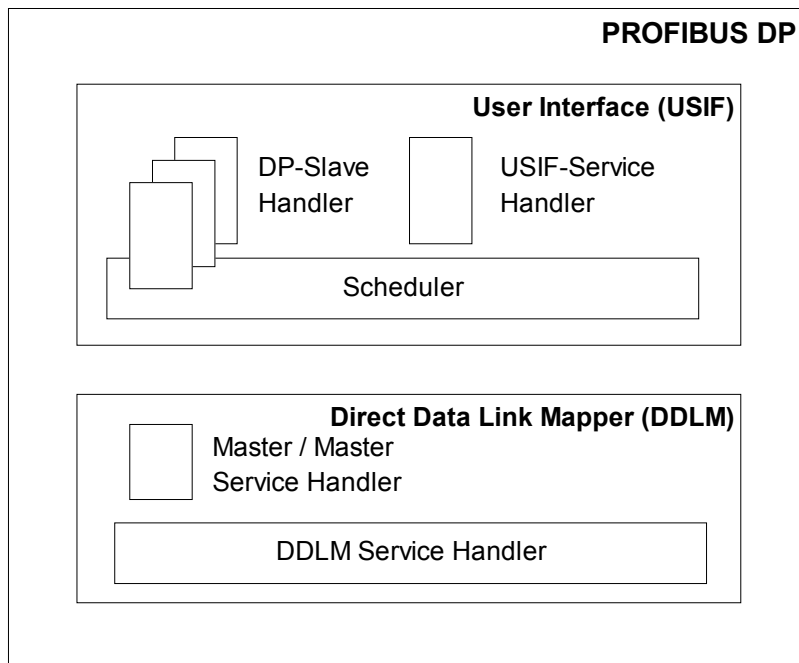


Figure 3-1: PROFIBUS DP main components

The following descriptions refer to the "real" PROFIBUS DP functionality and deal with the User Interface (USIF) and the services to influence it.

## 3.2 BASIC STATE MACHINE

The state of the PROFIBUS DP software (USIF state) depends on the user application commands and partly on the process status of the activated DP Slaves.

The following figure depicts the basic states of the DP protocol stack and the appropriate services to influence them:

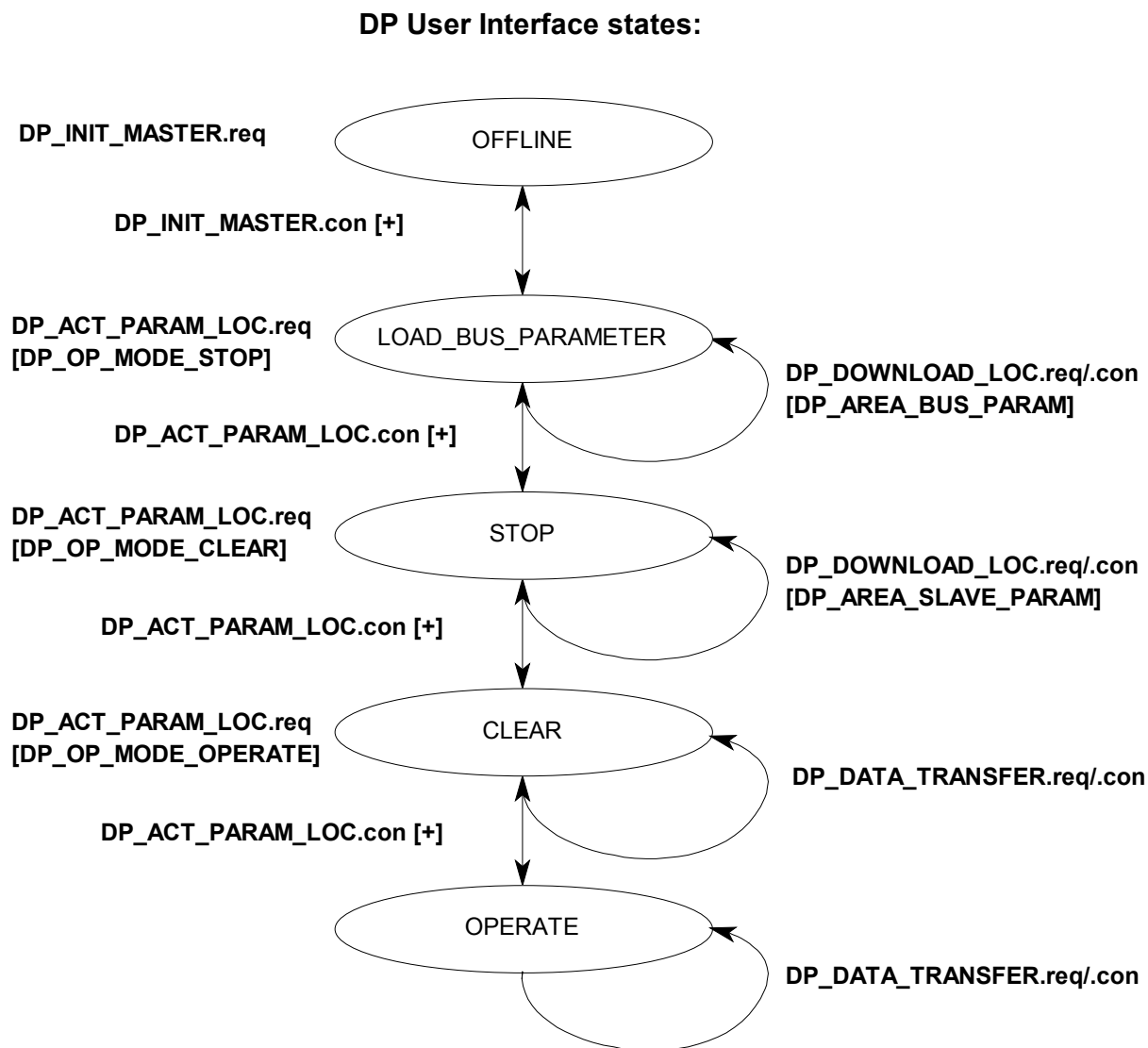


Figure 3-2: Main states of the DP protocol software (User Interface states)

### 3.3 SOFTWARE CONFIGURATION

The amount of memory that will be used during the PROFIBUS DP operation phase can be influenced by the user. The startup procedure is divided into two parts:

- **init\_profi function:**

Sets up the interface structures and creates the service and data interface. This function is described in the "User Interface" part of the PROFIBUS User Manual.

- **init\_master service:**

Determines the amount of resources that can be used within the DP protocol stack out of the global pool of available resources (for FMS, DP, FDL).

The "init\_master" service is also used to define the:

- Address Assignment Mode (AAM) for DP Slave I/O data location
- default Master station address and name for the default Bus Parameter Set
- operation mode of Service Access Point 54 as requester (M2) or responder (M1)
- handling of status information and I/O areas

These features will be explained in the following paragraphs and in the service interface description.

The following options are determined via compile time variables and are not changeable by the user:

- starting address of the DPRAM area
- DP Slave I/O data destination is DPRAM or RAM (direct copying of DP Slave I/O data by means of hardware support, e.g. ASPC2 (Advanced SIEMENS Profibus Controller 2))
- amount of DPRAM that can be used for I/O channels
- support of Status Information within the DPRAM
- support of timer to control the duration of different DP intervals or service durations

The amount of resources managed by the PROFIBUS DP communication software is limited with regard to processing speed and efficient memory handling (combi stacks, FDL access, process requirements). By means of the first service DP\_INIT\_MASTER a fixed number of memory blocks will be allocated. These resources cannot be changed during the PROFIBUS DP operation phase. Reconfiguration may only occur within the limits of that predefined memory amount. If the user enters the USIF-state OFFLINE again all memory blocks are released and the startup procedure can be repeated.

### 3.4 DP MASTER PARAMETER SET

The Master Parameter Set consists of:

- one Bus Parameter Set
- max. 126 DP Slave Parameter Sets ("max\_number\_slaves" defined in "FMB\_Set\_Configuration")

All parameter sets are transferred between DP User application and DP protocol stack by means of the services DP\_DOWNLOAD\_LOC, DP\_UPLOAD\_LOC and DP\_GET\_SLAVE\_PARAM.

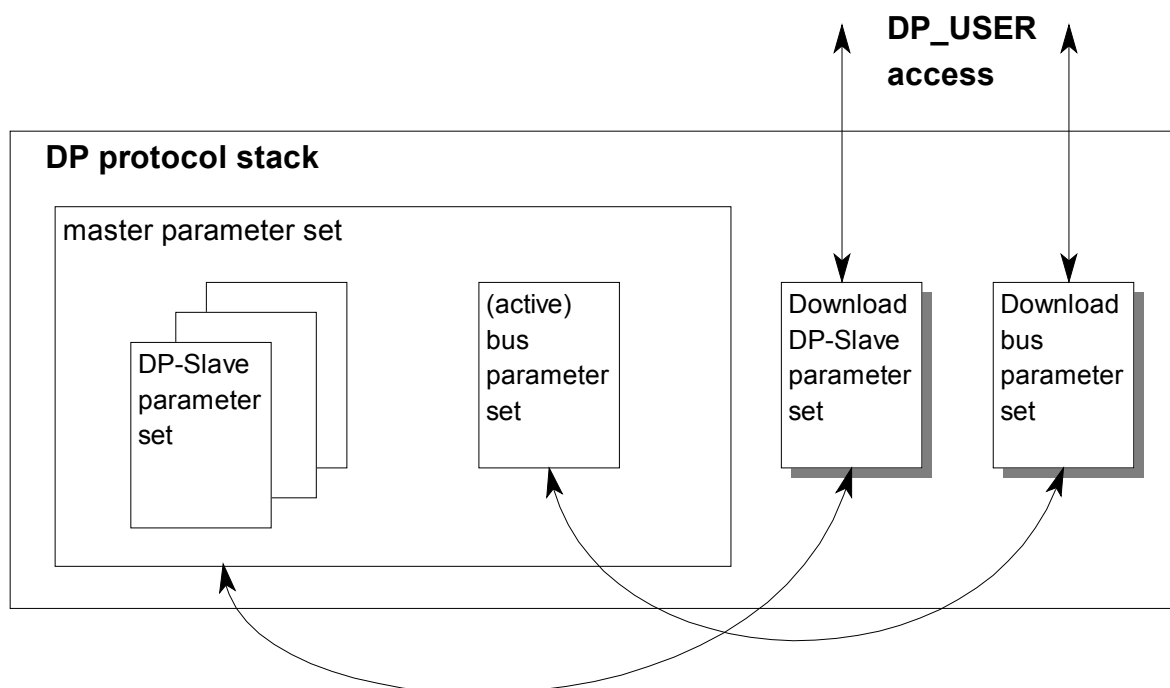


Figure 3-3: PROFIBUS DP Master Parameter Set

The DP Bus Parameter Set consists of the standard FDL operational parameters and some new DP specific extensions. The following structure shows the parameters in detail:

## Data Structure Bus Parameter Set:

Data Structure	T_DP_BUS_PARA_SET	
USIGN16	bus_para_len	DP_MIN_BUS_PARA_LEN.. max_bus_para_len (see FMB_SET_CONFIGURATION)
USIGN8	fdl_add	0..125
USIGN8	baud_rate	code number <sup>1</sup>
USIGN16	tsl	see EN 50170/2 (FDL)
USIGN16	min_tsdr	
USIGN16	max_tsdr	
USIGN8	tqui	
USIGN8	tset	
USIGN32	ttr	
USIGN8	g	
USIGN8	hsa	
USIGN8	max_retry_limit	
USIGN8	bp_flag	User Interface flags <sup>2</sup>
USIGN16	min_slave_intervall	$1..2^{16}-1$ , [100µs]
USIGN16	poll_timeout	$1..2^{16}-1$ , [1ms]
USIGN16	data_control_time	$1..2^{16}-1$ , [10ms]
OCTET	reserved [6]	
USIGN16	master_user_data_len	DP_MASTER_USER_DATA_LEN.. $2^{16}-33$
STRINGV	master_class2_name [32]	master who created the parameter set
OCTET	master_user_data [master_user_data_len - 34]	

---

<sup>1</sup>	0	DP_KBAUD_9_6
	1	DP_KBAUD_19_2
	2	DP_KBAUD_93_75
	3	DP_KBAUD_187_5
	4	DP_KBAUD_500
	5	DP_KBAUD_RESERVED
	6	DP_KBAUD_1500
	6	DP_MBAUD_1_5
	7	DP_KBAUD_3000
	7	DP_MBAUD_3
	8	DP_KBAUD_6000
	8	DP_MBAUD_6
	9	DP_KBAUD_12000
	9	DP_MBAUD_12
	10	reserved
	11	DP_KBAUD_45_45
	12..255	reserved for future baud rates

<sup>2</sup>	Bit 7	DP_BP_ERROR_ACTION
	Bits 6..0	reserved (cleared)

### 3.5 DP SLAVE PARAMETER SETS

The DP Slave parameter sets are used to declare the features of each DP Slave and to define the whole DP application. It is the most important data base for the user.

The structure of the Slave Parameter Sets is exactly the same defined in the PROFIBUS DP standard.

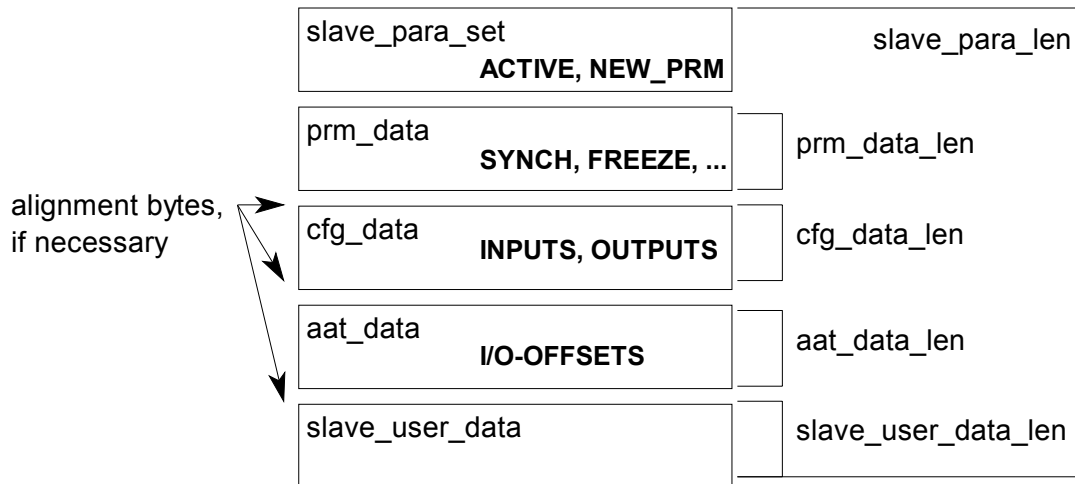


Figure 3-4: DP Slave Parameter Set layout

The Slave Parameter Sets are always set up with the byte ordering used by Motorola processors (high byte first). If data alignment to **WORD** or **LONG** variables is needed (e.g. by a host computer) additional alignment bytes may be inserted as shown above. That means if for example the **prm\_data\_len** is of an odd length one additional alignment byte may be inserted. The following **cfg\_data** then starts on an even address.

The DP protocol stack will recognize the alignment, and if no alignment has been provided the data set will be aligned automatically. So the user can operate either with "standard" DP Slave data sets or with aligned data sets.

For a detailed explanation of any parameter in the parameter sets see also the standard EN 50170/2 (FDL and DP). The structures for DP Slave parameter sets look like follows and are defined in PB\_DP.H:

*struct T\_DP\_SLAVE\_PARA\_SET*

USIGN16	slave_para_len	24..DP_MAX_SLAVE_PARA_LEN
USIGN8	sl_flag	DP_SL_ACTIVE, DP_SL_FLAG_NEW_PRM
USIGN8	slave_type	DP_SLAVE_TYPE_DP
OCTET	reserved [12]	
T_DP_PRM_DATA	prm_data	
T_DP_CFG_DATA	cfg_data	
T_DP_AAT_DATA	aat_data	
T_DP_USER_DATA	user_data	

**struct T\_DP\_PRM\_DATA**

USIGN16	prm_data_len	9..DP_MAX_PRM_DATA_LEN (high, low)
OCTET	station_status	DP_PRM_xxx
OCTET	wd_fact_1	watch dog factors
OCTET	wd_fact_2	t [s] = 10[ms]*wd_fact_1*wd_fact_2
OCTET	min_tsdr	t[bit]
USIGN16	ident_number	PNO ident number
OCTET	group_ident	group member bits
OCTET	prm_user_data [0..DP_MAX_USER_PRM_DATA_LEN]	

**struct T\_DP\_CFG\_DATA**

USIGN16	cfg_data_len	2..DP_MAX_CFG_DATA_LEN
OCTET	cfg_data [cfg_data_len - 2]	see EN 50170/2 (DP), DDLM_Chk_Cfg

**struct T\_DP\_AAT\_DATA**

USIGN16	aat_data_len	2..DP_MAX_AAT_DATA_LEN (high, low)
USIGN8	number_inputs	must be identically with cfg_data (in bytes)
USIGN8	number_outputs	
USIGN16	offset_inputs [1..number_inputs]	see next chapter for explanation
USIGN16	offset_outputs [1..number_outputs]	

**struct T\_DP\_SLAVE\_USER\_DATA**

USIGN16	slave_user_data_len	2..DP_MAX_SLAVE_USER_DATA_LEN
OCTET	slave_user_data [slave_user_data_len - 2]	

### 3.6 DPRAM ADDRESS ASSIGNMENT MODES

Since user applications always have different requirements in terms of communication participants, I/O data memory consumption or DP Slave I/O address assignment to locations within the DPRAM, appropriate addressing modes will be provided.

That means the user may decide where the I/O data of each DP Slave is copied to. So the user is able to adapt the fieldbus application to a specified automation task (e.g. in PLC programming). During DP Master initialisation the user must decide which Address Assignment Mode (AAM) should be used:

- DP\_AAM\_ARRAY
- DP\_AAM\_DEFINED
- DP\_AAM\_COMPACT
- DP\_AAM\_IO-BLOCK

By means of an Address Assignment Table (AAT) within the Slave Parameter Sets the RAM/DPRAM layout can be defined. Not each Address Assignment Mode requires the definition of an Address Assignment Table. In the mode "ARRAY" an AAT is not needed! The configuration data of each DP Slave will be evaluated therefore.



### 3.6.1 "ARRAY" Mode

The *Array* I/O Address Assignment Mode is intended for very easy use without concerning any address location requirements. This mode is the default AAM. The AAT is not used or ignored.

The DP Slave input and output data will be mapped contiguously into the RAM/DPRAM. The I/O memory areas are located in order of ascending station addresses and have always the same length.

That means if the station addresses are not contiguous gaps will occur between the DP Slave I/O memory areas. The number of stations to be supported depends on the size of DPRAM and the maximum length of inputs and outputs.

Following parameters influence the address assignment and are determined by means of the `DP_INIT_MASTER` service:

- `LOWEST_SLAVE_ADDRESS`: address of the first DP Slave station to be located at the beginning of the DPRAM I/O area
- `MAX_NUMBER_SLAVES`: maximum number of DP Slaves supported
- `MAX_SLAVE_INPUT_LEN`: maximum length of DP Slave input values
- `MAX_SLAVE_OUTPUT_LEN`: maximum length of DP Slave output values

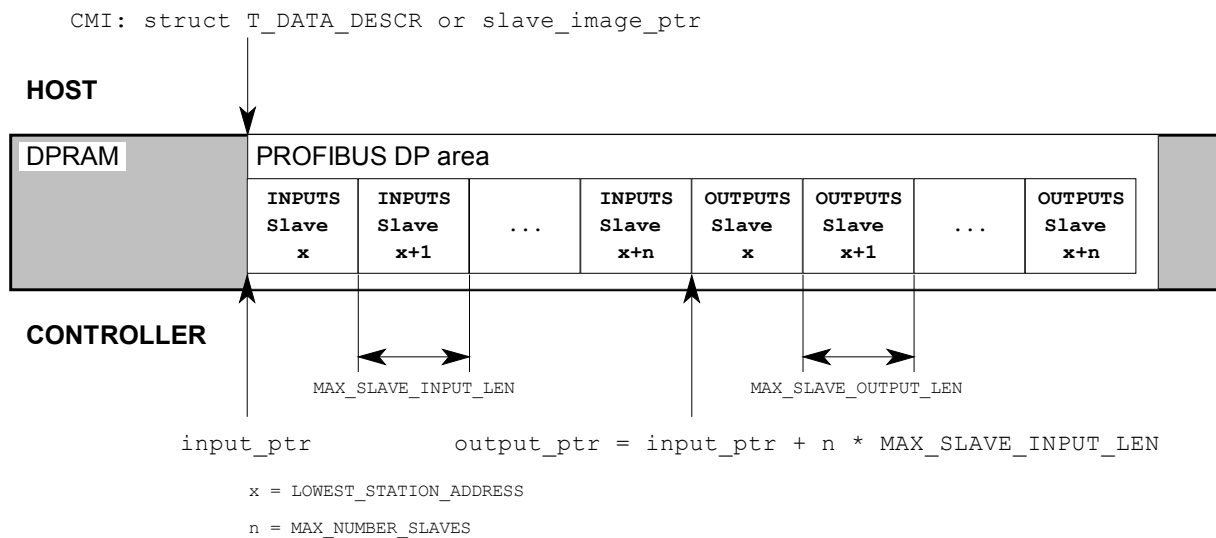


Figure 3-5: DP Slave I/O data in Array Address Assignment Mode

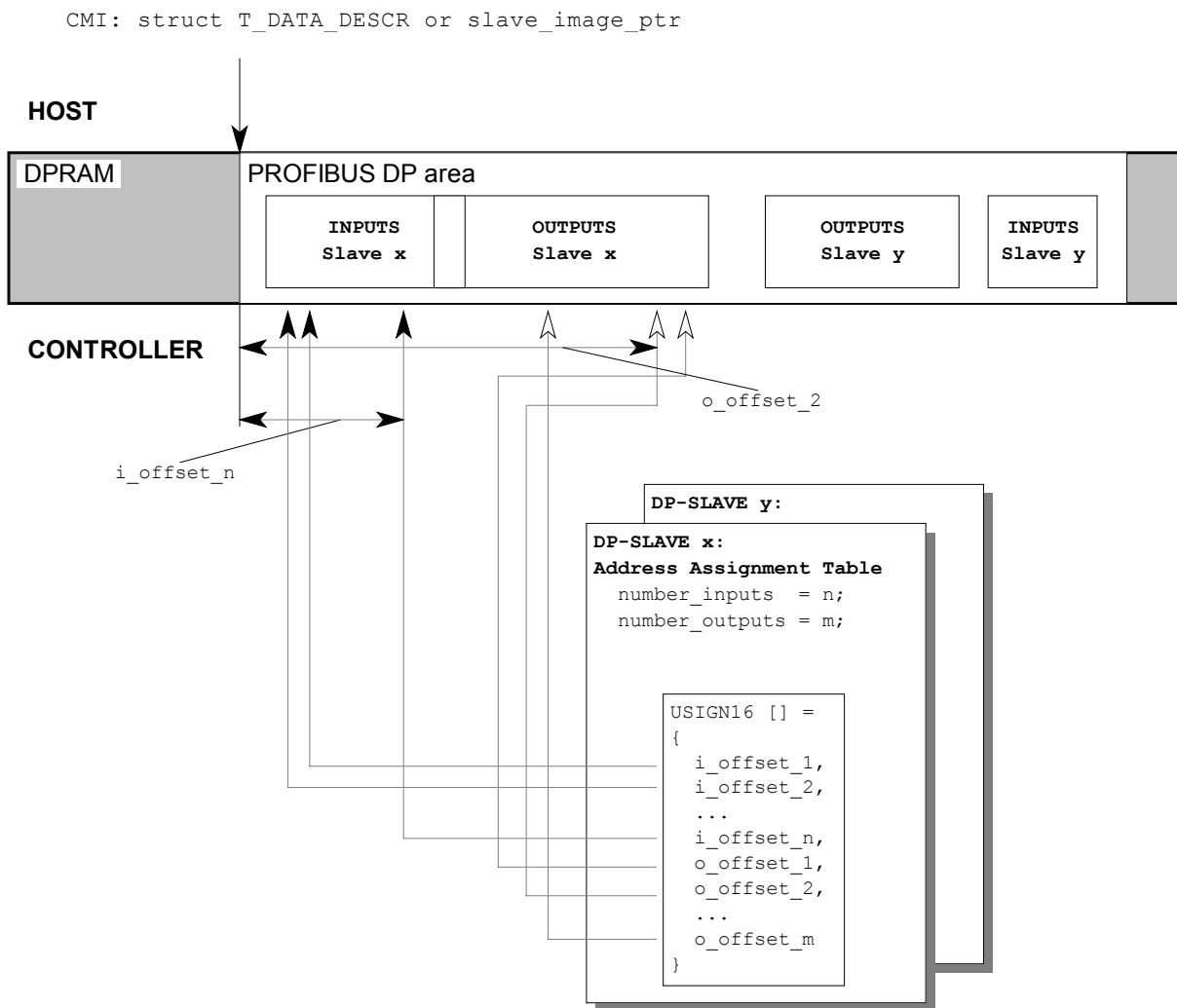
**Note, the Array I/O Address Assignment Mode cannot be used together with the PROFIBUS Windows-NT driver.**

## 3.6.2 "USER DEFINED" Mode

Often applications expect special memory layouts to fit their internal addressing (e.g. PLC's). The *User Defined* Address Assignment Mode offers the opportunity to determine the location of each input / output byte of a DP Slave within a 64 Kbytes segment.

The location of each input and output byte must be defined by one offset value per byte. Thus the memory area is fragmented and the user is responsible for the memory layout (overlapping areas etc.).

The following figure depicts the address assignment:



**Note, the User Defined I/O Address Assignment Mode cannot be used together with the PROFIBUS Windows-NT driver.**

### 3.6.3 "COMPACT" Mode

The *Compact* I/O Address Assignment Mode combines several features of modes "Array" and "User Defined" to find a compromise in terms of easy use and low memory consumption.

Definition of an Address Assignment Table (AAM) within the Slave Parameter Set is necessary and will be checked by the protocol software. The user is responsible for defining one starting offset for the input and the output region of the respective DP Slave. That means overlapping areas are not detected by the protocol stack! The starting offsets must be even values.

The I/O bytes are always located contiguously so that the linear byte access may be used. The following figure depicts the Compact Address Assignment Mode:

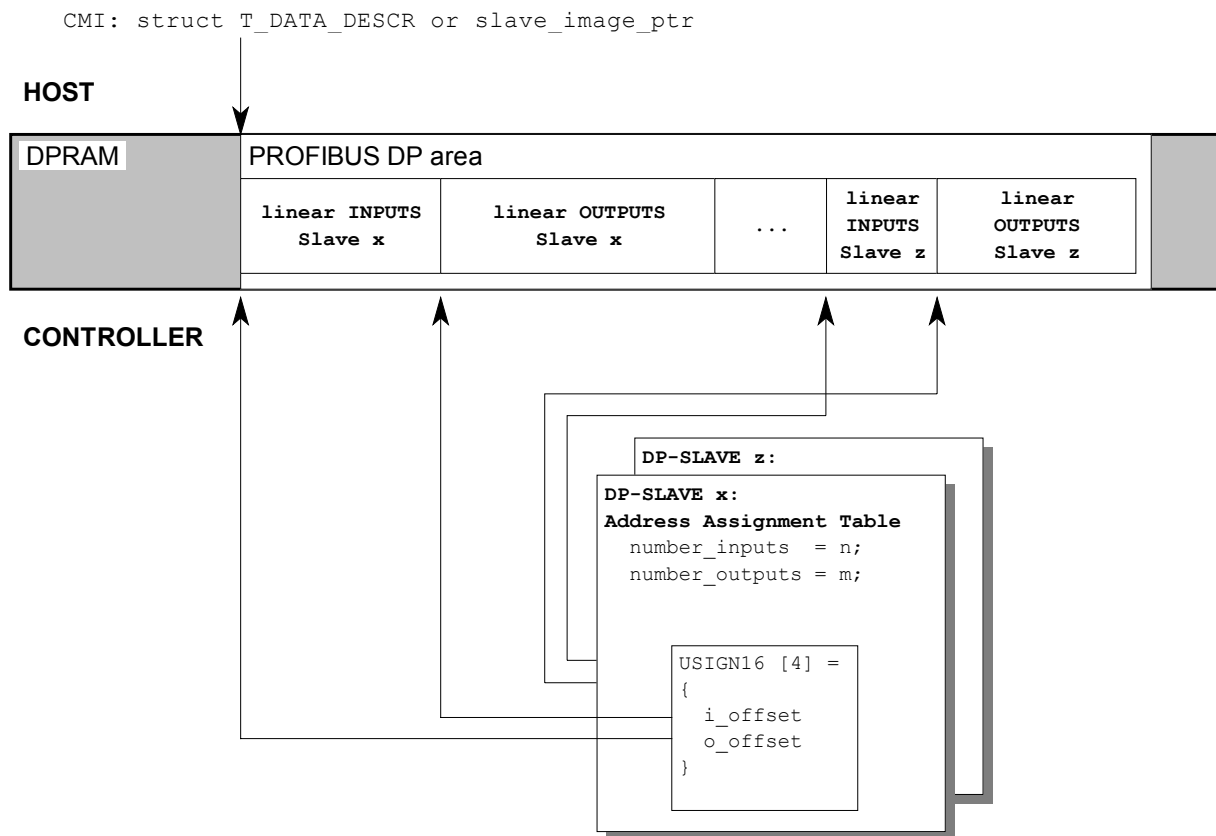


Figure 3-7: DP Slave I/O data in Compact Address Assignment Mode

**Note, the Compact I/O Address Assignment Mode cannot be used together with the PROFIBUS Windows-NT driver.**

### **3.6.4 "IO-BLOCK" Mode**

The DP protocol software and the PROFIBUS Windows NT driver use a special handshake mechanism to provide the high performance data interface for DP Slaves. This requires a special data ordering and alignment within the Dual Ported RAM interface.

The DP protocol software therefore provides the new Address Assignment Mode (AAM) called IO\_BLOCKS which is designed to operate hand in hand with the PROFIBUS Windows NT driver.

Note, the AAM *IO-BLOCK* is set by the PROFIBUS Windows NT driver automatically during DP\_Init\_Master service.

The AAM IO\_BLOCKS is handled by the DP Master automatically. That means, there is no need to configure any offsets to locate DP Slaves.

During the download procedure at the startup phase the DP Master locates each slave within the DPRAM. Once the DP Master has been forced to enter the state STOP this DPRAM layout is fixed.

Note, because the AAM IO\_BLOCKS uses a fixed DPRAM layout it is not possible to change the slave configuration dynamically! Thus, the service Download will return the error code "access denied" if the service is executed after entering the state STOP.

#### **Example service sequence for DP initialization:**

Startup phase:

- DP\_Init\_Master (AAM\_IO\_BLOCKS, ...)
- DP\_Download\_Loc (Slave n)
- ...
- DP\_Download\_Loc (Slave m)
- DP\_Act\_Param\_Loc (STOP) ⇒ this service locks the DPRAM layout

Operation phase:

- DP\_Act\_Param\_Loc (CLEAR, OPERATE)
- other DP services

By means of the service DP\_Exit\_Master it is possible to return to the startup phase of the configuration. Thus, it is possible to reconfigure all DP Slaves and to relayout the DPRAM after the service DP\_Exit\_Master.

## How to find out the real position of a DP Slave in the DPRAM using the compatibility mode of the Windows NT PROFIBUS API?

Since the DP Master automatically locates the DP Slaves in the DPRAM it is necessary to find out the real position of the slave I/O data. Therefore the service DP\_Get\_Slave\_Param should be used:

Example:

- Startup phase as described above
- Then use the following service for each DP Slave:
  - DP\_Get\_Slave\_Param.req (identifier=DP\_SLAVE\_PARAM\_SLAVE\_INFO, rem\_add=slave\_address)
  - DP\_Get\_Slave\_Param.con (T\_DP\_SLAVE\_PARAM\_SLAVE\_INFO)

```
typedef struct _T_DP_SLAVE_PARAM_SLAVE_INFO
{
    USIGN16    diag_entries,          /* available DIAG messages */
    USIGN16    offset_inputs;         /* offset of input area within I/O memory */
    USIGN16    offset_outputs;        /* offset of output area within I/O memory */
    USIGN8     number_inputs;         /* inputs of I/O memory */
    USIGN8     number_outputs;        /* outputs of I/O memory */
    USIGN8     sl_flag;               /* DP_SL_ACTIVE, DP_SL_FLAG_NEW_PRM */
    USIGN8     slave_type;            /* DP_SLAVE_TYPE_DP */
} T_DP_SLAVE_PARAM_SLAVE_INFO;
```

- The parameters offset\_inputs and offset\_outputs define the position of the respective DP Slave related to the beginning of the DPRAM area of slave I/O data. These parameters are used in the services profi\_set\_data and profi\_get\_data to access the I/O memory of DP Slaves.

### 3.7 LOCAL / REMOTE SERVICES

Every service issued by a DP Master (class 2) has a local counterpart. That means the data structures used for local and remote services are always identical.

Via the `INIT_MASTER` service the DP user application defines how the DP Master (class 1) reacts when receiving a remote service request. The parameter `auto_remote_services` controls the behaviour of the Master as follows.

#### DP-User acknowledged remote services (`auto_remote_services = FALSE`):

In this configuration mode the DP-User application is responsible for acknowledging remote service requests. The advantage is that the DP application can always update its local data base when receiving a remote service request.

The DP-User application that receives a remote service indication must use the respective local service to handle the request. After receiving the local confirmation the application must send the response frame to the remote DP Master (class 2). The following figure depicts the mapping of remote services onto the local ones by means of the `ACT_PARAM` service:

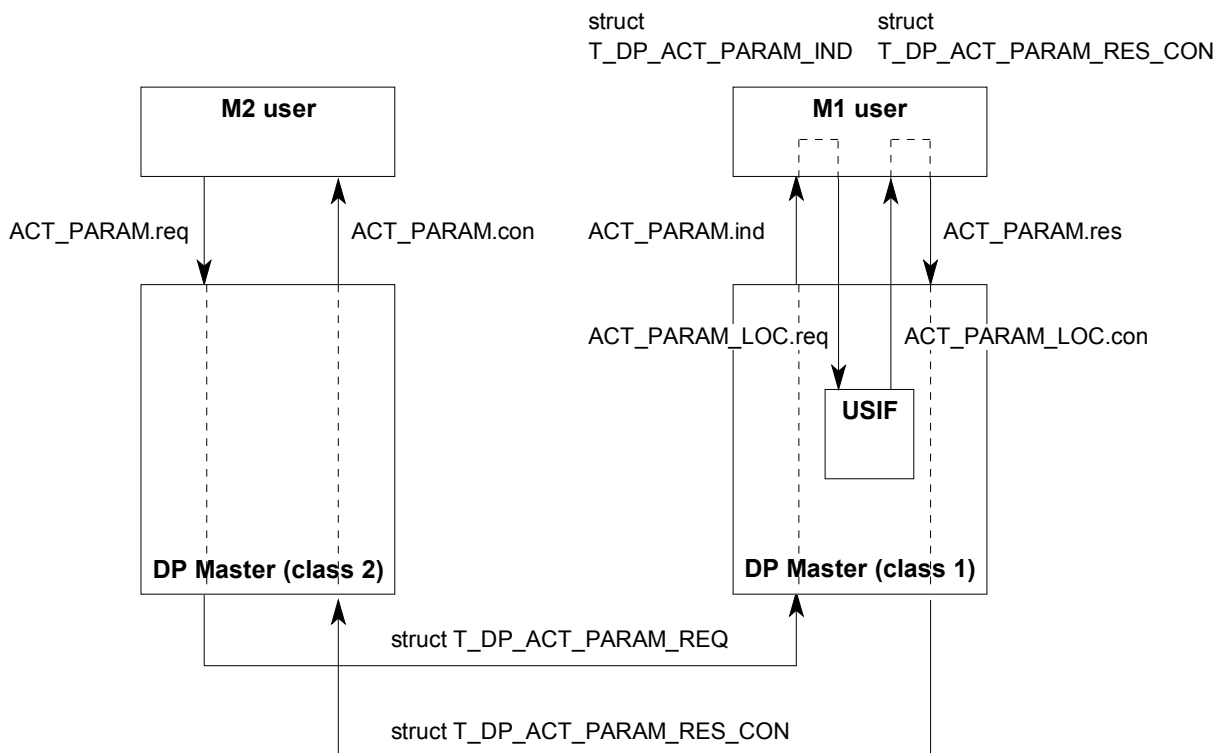


Figure 3-8: Mapping of local and remote DP Master / Master services by the DP User application

### DP Master acknowledged remote services (auto\_remote\_services = TRUE)

In this configuration mode the DP Master (class 1) answers to all incoming remote services automatically. That means the DP-User application does not have to react on Master (class 2) service requests! Thus the application also cannot update its local data structures if the configuration has been changed by a DP Master (class 2). The following figure illustrates this mechanism:

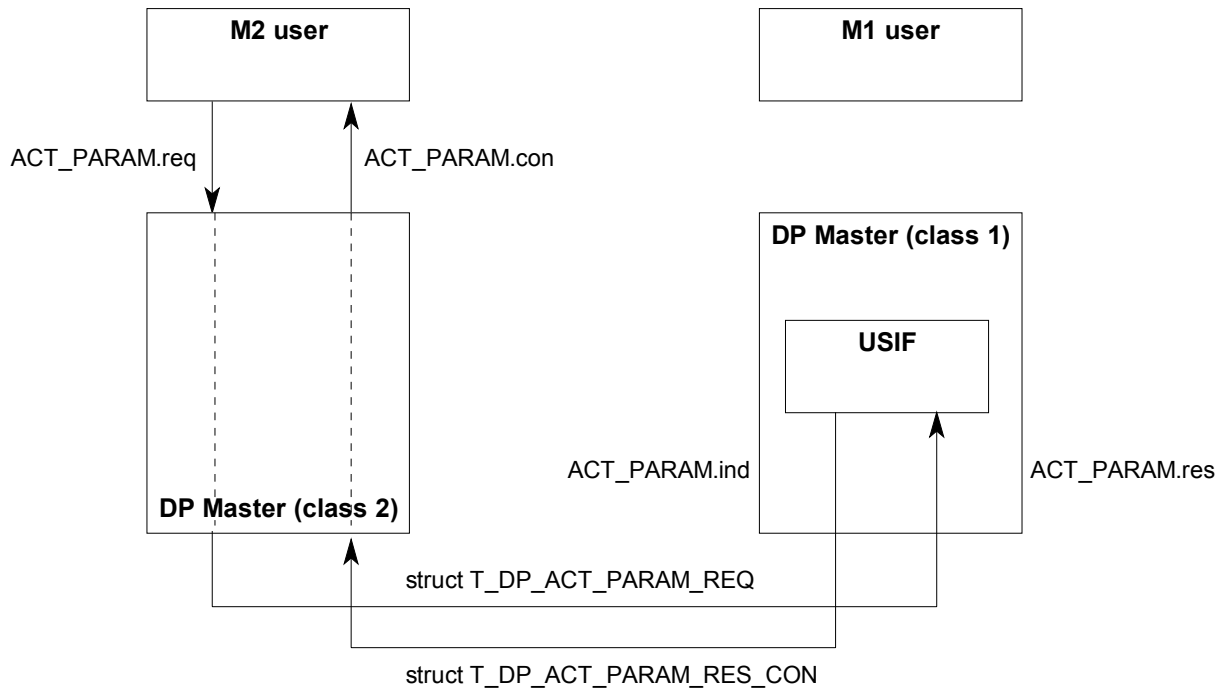


Figure 3-9: Automatic response to Master / Master services by the DP protocol software

## 4 SERVICE INTERFACE

This chapter describes the functionality and syntax formats of service calls needed to run a user application on top of the PROFIBUS DP protocol stack. Therefore the content of all necessary data structures is listed.

The service explanations are structured concerning user requirements. For example some services of the DDLM sublayer are only used by DP Master (class 2) users and of no interest for usual PROFIBUS DP applications. That is why those services are explained at the end of the interface description.

All services visible to the user are implemented with regards to EN 50170/2 (DP).

### Service return values

Every PROFIBUS DP service returns two bytes status information at the beginning of the service data area. The values and constants of the error codes are defined in the `PB_ERR.H` header file.

This status value is defined as follows:

- status low byte: contains the error code like `OK`, `IV`, `NO`, ...
- status high byte: contains the reason code if available (not zero) like `E_DP_WRONG_SLAVE_ADD`, `E_DP_DATA_ALIGNMENT`, ...

### 4.1 OVERVIEW

The following sections show all available DP services arranged in functional groups together with a short description of their method of working.

Initialization / Termination	Description
<code>DP_INIT_MASTER</code>	configures the protocol stack: - operation mode (Master 1/2) - Slave I/O data layout
<code>DP_EXIT_MASTER</code>	terminates the protocol stack: - release all activated DP Slaves - free dynamic memory - stop the bus access (exit token ring)



Master Parameter Set Access	Description
DP_DOWNLOAD_LOC / DP_DOWNLOAD	transfers Slave or Bus Parameter Sets: - DP-User application ⇔ DP Master (local) - DP-Configurator ⇔ DP Master (remote)
DP_UPLOAD_LOC / DP_UPLOAD	transfers Slave or Bus Parameter Sets: - DP Master (local) ⇔ DP-User application - DP Master (remote) ⇔ DP-Configurator
DP_START_SEQ_LOC / DP_START_SEQ DP_END_SEQ_LOC / DP_END_SEQ	access protection during up/downloads: - no local / remote access to the specified parameter set - allows multiple up / downloads by means of data blocks and offsets
DP_SET_PRM_LOC	sets new Slave parameters during the data exchange phase
DP_GET_SLAVE_PARAM	individual upload of Slave Parameter Set sections (header, prm, cfg, aat, user data)

Status / Operation Control	Description
DP_ACT_PARAM_LOC / DP_ACT_PARAM	controls the status of DP Masters and Slaves: - operation mode setting (OFFLINE... OPERATE) - (de)activating DP Slaves - activating new bus parameters (local)
DP_ACT_PARA_BRCT	activates new bus parameters within the whole PROFIBUS DP network
DP_DATA_TRANSFER	synchronises the DP-User application with the DP Master polling cycles
DP_GLOBAL_CONTROL	sends controlling commands to DP Slaves or groups of Slaves
DP_SET_SLAVE_ADD	assigns new station addresses to DP Slaves across the DP network

<b>Diagnosis / Information</b>	<b>Description</b>
DP_GET_SLAVE_DIAG	provides diagnostic messages of DP Slaves out of the internal circular buffer of the DP Master (class 1)
DP_GET_MASTER_DIAG_LOC / DP_GET_MASTER_DIAG	supplies status information about the DP Master and related Slaves: - current Slave diagnostic information - Master operation mode - system diagnostic data - data transfer list
DP_GET_SLAVE_PARAM	local information service: - Slave specific information - system status information
DP_RD_INP / DP_RD_OUTP	acyclic read of DP Slave inputs / outputs for diagnostic purposes
DP_GET_CFG	reads the "real" config data of DP Slaves

Note: The following services may conflict with the local DP Master (class 1) operation and are intended for test and configuration purposes!

<b>Direct DDLM Access</b>	<b>Description</b>
DP_SET_PRM	sends new Slave parameters to DP Slaves: - set operation mode (sync, freeze, ...) - change Min Tsdr
DP_CHK_CFG	compares configuration data with the "real" CFG-Data of DP Slaves
DP_SLAVE_DIAG	reads the current diagnostic status from a DP Slave
DP_DATA_EXCHANGE	transmits output data to DP Slaves and reads input data

## 4.2 INITIALIZATION / TERMINATION

### 4.2.1 Init\_Master

This service is provided to allow adaptation of manufacturer and process specific parameters to the PROFIBUS DP protocol stack and define the operation modes.

With consideration to an easy and effective memory management in combination with PROFIBUS FMS the amount of memory that might be used for each stack can be different. That is why several maximum values for parameters must be assigned by means of this service. User specific reconfiguration later must only take place within the preset maximum limits (e.g. download of Slave Parameter Sets).

The service must be called before any other PROFIBUS DP service!

After resetting the protocol stack to the "OFFLINE" state (service: ACT\_PARAM\_LOC) the initialisation may be repeated. The User Interface enters the "LOAD\_BUS PARA" state if the service was successful.

#### *Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_INIT_MASTER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for local Request:*

Data Structure	T_DP_INIT_MASTER_REQ	
USIGN8	master_default_address	0..125, default master station address
PB_BOOL	master_class2	TRUE: SAP 54 acts as requester (M2) FALSE: SAP 54 acts as responder (M1)
STRINGV	master_class2_name [32]	M2 name for the default Bus Parameter Set
USIGN8	lowest_slave_address	0.. DP_MAX_SLAVE_ADDRESS
USIGN8	slave_io_address_mode	Address Assignment Mode <sup>3</sup>
PB_BOOL	clear_outputs	TRUE: in mode "CLEAR" outputs will be set to zero before starting the polling cycle
USIGN8	auto_remote_services	Bit mask to determine which M1 responses to M2 requests should be acknowledged automatically by the protocol stack <sup>4</sup>
PB_BOOL	cyclic_data_transfer	TRUE: when entering the state "CLEAR" or "OPERATE" the polling cycles start without service "DATA_TRANSFER"

---

<sup>3</sup>	00H	DP_AAM_ARRAY
	01H	DP_AAM_DEFINED
	02H	DP_AAM_COMPACT

<sup>4</sup>	80H	DP_AUTO_GET_MASTER_DIAG
	40H	DP_AUTO_UPLOAD_DOWNLOAD_SEQ
	20H	DP_AUTO_ACT_PARAM
	E0H	DP_AUTO_REMOTE_SERVICES
	00H	DP_USER_REMOTE_SERVICES

**Note:** If the user directs the protocol stack to go directly in the "STOP" state after initialisation then the default Bus Parameter Set will be activated. The parameters "master\_default\_address" and "master\_class2\_name" will be used in this parameter set.

The following parameters determine the behaviour of the DP Master:

- **master\_class2:** If this option is enabled the DP Master operates as DP Master (class 2), i.e. as parameterization and configuration device otherwise as service responder
- **auto\_remote\_services:** If the DP Master is configured as M2 responder (`master_class2 = FALSE`) and this option is activated for the respective service then incoming remote services will be answered automatically by the Master and not indicated to the user application! That means the user has not to provide any M2 handling routines. (see also chapter "3.7 Local / Remote Services" for explanation)
- **cyclic\_data\_transfer:** If there is no need to synchronise the DP user application with the polling cycles of the DP Master this option should be activated. The Master will automatically start polling cycles to DP Slaves when entering the "CLEAR" or "OPERATE" state.

Note, that there is only data consistency guaranteed when accessing the DP Slave I/O areas by means of the Softing PROFIBUS Application Program Interface services. The DP Master indicates new diagnostic messages or operation mode changes with the services `GET_SLAVE_DIAG.ind` and `ACT_PARAM_LOC.ind` to the DP user application. That means the user application does not have to poll diagnostic messages.

## *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_INIT_MASTER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

## *Data Block for local Confirmation:*

Data Structure	T_DP_INIT_MASTER_CON	
USIGN16	status	OK, IV, NO

## 4.2.2 Exit\_Master

This service is the termination counterpart of the "Init\_Master" service. It should be used to:

- release all activated DP Slaves and make them available again for other DP Masters,
- free the internal dynamic memory,
- disconnect the Master from the PROFIBUS (stop the participation in the token protocol).

The service is independent on the Master's operation state (STOP, CLEAR, OPERATE) but not executable within the OFFLINE state.

### *Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_EXIT_MASTER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### *Data Block for local Request:*

VOID

### *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_EXIT_MASTER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

### *Data Block for local Confirmation:*

Data Structure	T_DP_EXIT_MASTER_CON	
USIGN16	status	OK, NO

### **4.3 DP MASTER (CLASS 1) SERVICE INTERFACE**

The DP Master (class 1) contains a predefined fieldbus application, the so called User Interface. It is responsible for parameterization, configuration of DP Slaves and the data exchange with them.

During startup several initialisation actions have to be done to prepare the stand alone cyclic data exchange of the User Interface to all assigned DP Slaves (USIF-states "CLEAR/OPERATE").

This chapter provides all information necessary to build up a fieldbus application acting as DP Master (class 1) user onto the PROFIBUS DP protocol stack.

See also chapter "Basic state machine" in this documentation.

The following services defined in EN 50170/2 (DP) have been replaced with the respective local services to get a uniform service interface:

- Set\_Mode ⇔ Act\_Param\_Loc
- Mode\_Changed ⇔ Act\_Param\_Loc
- Load\_Bus\_Par ⇔ Download\_Loc
- Delete\_SC ⇔ Download\_Loc (not supported yet)

### 4.3.1 Upload\_Loc / Download\_Loc

The upload / download services can be used by the local user of a DP Master (class 1) to:

- transfer or clear DP Slave parameter sets
- transfer the DP Master Bus Parameter Set

The data structures are identical to those used in the remote services ("upload/download").

Downloads with a data length of zero mean to delete the addressed data area (i.e. Slave Parameter Sets).

In extension to the PROFIBUS DP standard two ways to download data areas have been implemented. The difference is the automatic parameter check by the protocol stack at the end of a download cycle. That means a definite download period is needed to realise a complete area access (usually there is no termination sign after a download sequence).

#### Start\_Seq, Download, End\_Seq

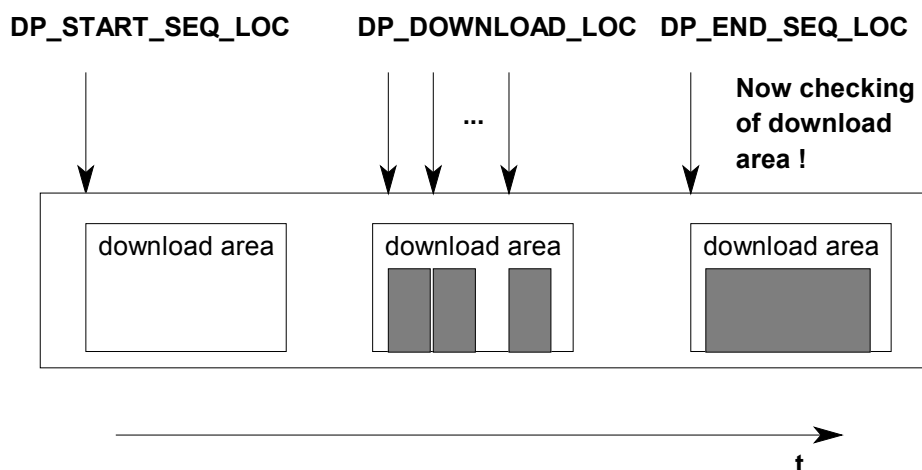


Figure 4-1: Download sequence with area check

#### Single Download

To ease the implementation of the standard case with data length not longer as the maximum PROFIBUS telegram length the second download method is provided.

Each download will be treated as complete cycle and parameter checking will take place immediately.

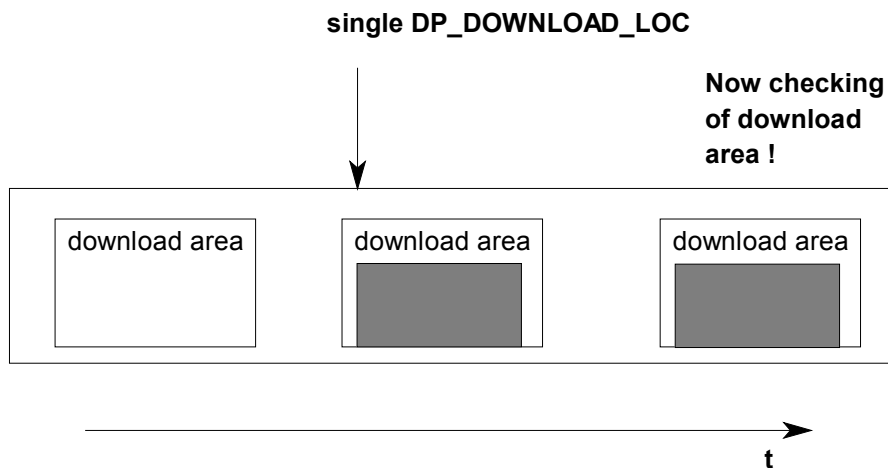


Figure 4-2: Single Download with area check (no sequence commands)

## Download\_Loc

### Service Description Block for local Request:

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_DOWNLOAD_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### Data Block for local Request:

Data Structure	T_DP_DOWNLOAD_REQ	
USIGN16	data_len	0..DP_MAX_DOWNLOAD_DATA_LEN
USIGN8	rem_add	M1 station address
USIGN8	area_code	destination area <sup>5</sup>
USIGN16	add_offset	data block offset
OCTET	data [data_len]	

### Service Description Block for local Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_DOWNLOAD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

<sup>5</sup>

0..125	DP_AREA_SLAVE_PARAM (Slave parameter set of the specified DP-Slave)
127	DP_AREA_BUS_PARAM
129	DP_AREA_STAT_COUNT (not implemented yet)

all other values are reserved



*Data Block for local Confirmation:*

Data Structure	T_DP_DOWNLOAD_RES_CON	
USIGN16	status	OK, AD, EA, LE, SC, NC

## Upload\_Loc

*Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_UPLOAD_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

*Data Block for local Request:*

Data Structure	T_DP_UPLOAD_REQ	
USIGN8	rem_add	M1 station address
USIGN8	area_code	destination area (see download)
USIGN16	add_offset	data block starting offset
USIGN8	data_len	1..DP_MAX_UPLOAD_DATA_LEN
USIGN8	dummy	alignment byte

*Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_UPLOAD_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

*Data Block for local Confirmation:*

Data Structure	T_DP_UPLOAD_RES_CON	
USIGN16	status	OK, AD, EA, LE, SC, NC
USIGN16	data_len	0..DP_MAX_UPLOAD_DATA_LEN
OCTET	data [data_len]	

### 4.3.2 Start\_Seq\_Loc / End\_Seq\_Loc

By means of these services a sequence of "upload\_loc / download\_loc" is entered and terminated.

See the description and figures of "upload\_loc / download\_loc" in the previous section to get detailed information about service sequences.

Note, that in this implementation the services are not used to provide data access protection because remote service will be transferred to the user anyway. Furthermore the services marking the start and the end of a download sequence allow syntax checking of the downloaded data at the end.

#### Start\_Seq\_Loc

*Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_START_SEQ_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

*Data Block for local Request and Indication:*

Data Structure	T_DP_START_SEQ_REQ	
USIGN8	rem_add	M1 station address
USIGN8	area_code	destination area <sup>6</sup>
USIGN16	timeout	[1ms]

*Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_START_SEQ_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

*Data Block for local Confirmation:*

Data Structure	T_DP_START_SEQ_RES_CON	
USIGN16	status	OK, NO, SC
USIGN8	max_len_data_unit	1..DP_MAX_DOWNLOAD_DATA_LEN
USIGN8	dummy	alignment byte

#### End\_Seq\_Loc

*Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_END_SEQ_LOC

<sup>6</sup>

0..125	DP_AREA_SLAVE_PARAM (Slave parameter set of the specified DP-Slave)
127	DP_AREA_BUS_PARAM
129	DP_AREA_STAT_COUNT (not implemented yet)
255	no local access protection
all other values are reserved	

USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### *Data Block for local Request:*

Data Structure	T_DP_END_SEQ_REQ	
USIGN8	rem_add	M1 station address
USIGN8	dummy	alignment byte

### *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_END_SEQ_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

### *Data Block for local Confirmation:*

Data Structure	T_DP_END_SEQ_RES_CON	
USIGN16	status	OK, NO, SC

## 4.3.3 Act\_Param\_Loc

To control the status of the PROFIBUS DP protocol stack (User Interface) this service has to be used. The following settings can be influenced:

- activation / deactivation of DP Slaves (sets and resets the `DP_SL_ACTIVE` flag in the Slave Parameter Set)
- Bus Parameter Set activation
- operation mode selection of the PROFIBUS DP protocol stack (OFFLINE, STOP, CLEAR, OPERATE)

Indications inform about the following events:

- operation mode changes caused by DP Slave errors (i.e. AUTOCLEAR)
- operation mode changes caused by remote DP Masters (class 2)

### Service Description Block for local Request and Indication:

USIGN16	comm_ref	not used
USIGN8	layer	DP / DP_USR
USIGN8	service	DP_ACT_PARAM_LOC
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used
INT16	result	not used

### Data Block for local Request:

Data Structure                      T\_DP\_ACT\_PARAM\_REQ

USIGN8	rem_add	M1 station address
USIGN8	area_code	parameter set to be influenced <sup>7</sup>
USIGN8	activate	area_code dependent value <sup>8</sup>
USIGN8	dummy	alignment byte

<sup>7</sup>      0..125    DP\_AREA\_SLAVE\_PARAM (Slave parameter set of the specified DP-Slave)  
       127      DP\_AREA\_BUS\_PARAM    no baud rate changes!  
       128      DP\_AREA\_SET\_MODE  
       all other values are reserved

<sup>8</sup>      80H      DP\_SLAVE\_ACTIVATE (area\_code = DP\_AREA\_SLAVE\_PARAM)  
       00H      DP\_SLAVE\_DEACTIVATE  
       FFH      DP\_BUS\_PAR\_ACTIVATE (area\_code = DP\_AREA\_BUS\_PARAM)  
       00H      DP\_OP\_MODE\_OFFLINE (area\_code = DP\_AREA\_SET\_MODE)  
       40H      DP\_OP\_MODE\_STOP  
       80H      DP\_OP\_MODE\_CLEAR  
       C0H      DP\_OP\_MODE\_OPERATE

### *Data Block for local Indication:*

Data Structure	T_DP_ACT_PARAM_IND	
USIGN8	rem_add	M1 station address
USIGN8	area_code	see .req
USIGN8	activate	see .req
USIGN8	dummy	alignment byte

### *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_ACT_PARAM_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

### *Data Block for local Confirmation:*

Data Structure	T_DP_ACT_PARAM_RES_CON	
USIGN16	status	OK, NO, DS

## 4.3.4 Data\_Transfer

This service is provided to achieve synchronisation between the user and the DP protocol stack and to start DP Slave polling during the operation phase.

The service is very similar to the standard DP service "MARK". The most important difference is that the data transfer phases of the Softing DP protocol stack are not timer controlled in the configuration mode `cyclic_data_transfer=FALSE` (see service `INIT_MASTER`).

Each PROFIBUS DP polling cycle must be started by means of this service! The service must be called cyclically by the user when the protocol stack (USIF) has reached the state "CLEAR/OPERATE".

During execution of this service the DP Slave I/O areas are not consistent because incoming Slave data might be written directly to these areas. The confirmation indicates the end of Slave polling cycle. Diagnostic and Status Information now can be analysed.

The service confirmation informs about the number of diagnostic messages in the circular buffer. These messages may be obtained using the services `DP_GET_SLAVE_DIAG` (see the following chapter).

The following figure illustrates the use of `DP_DATA_TRANSFER`:

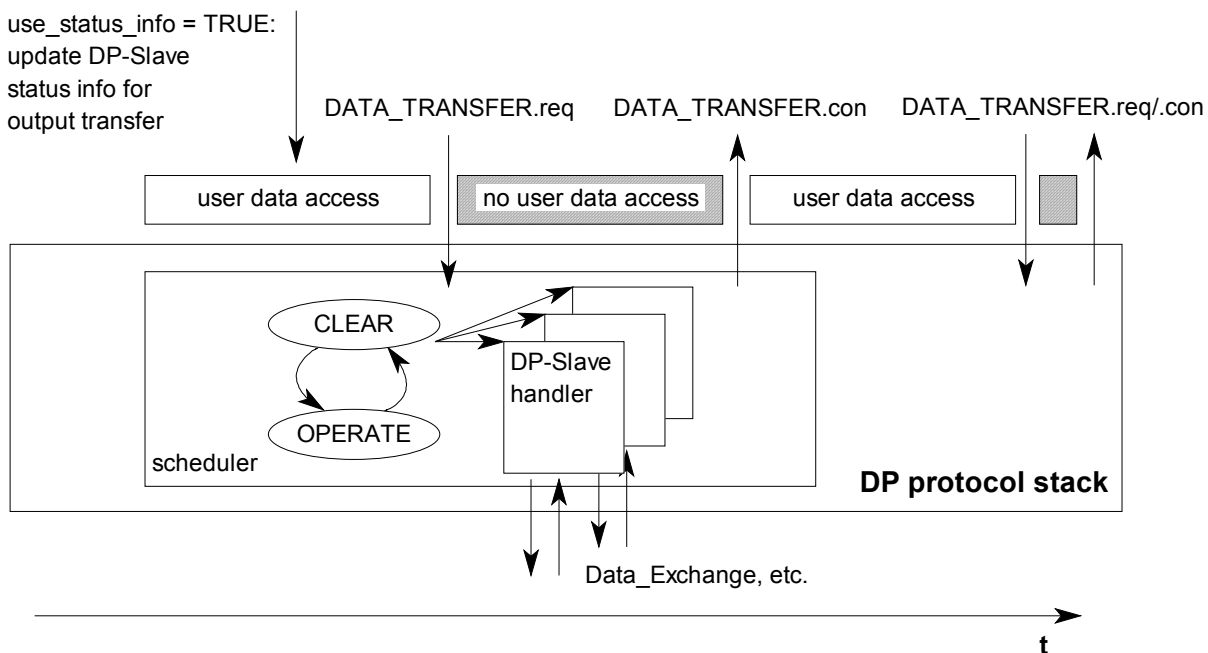


Figure 4-3: Service `DP_Data_Transfer` for activating polling cycles and synchronisation

*Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_DATA_TRANSFER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

*Data Block for local Request:*

VOID

*Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_DATA_TRANSFER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

*Data Block for local Confirmation:*

Data Structure	T_DP_DATA_TRANSFER_CON	
USIGN16	status	OK, NO
INT16	diag_entries	DP_SLAVE_DIAG_OVERFLOW.. max_slave_diag_entries (see "init_master")

### 4.3.5 Get\_Slave\_Diag

If a DP Slave indicates the presence of diagnostic data, the DP Master (class 1) will immediately call for these data. The diagnostic data will be entered in a circular buffer of parametrizable length (see service "init\_master"). Only if an overflow occurs the oldest entry will be overwritten and the overflow status flag will be set.

If an actual diagnostic image of any used DP Slave is needed, the DP-User application must set up this image by itself. Once initialised at startup every incoming diagnostic message must be used to update this area. The following figure illustrates this method:

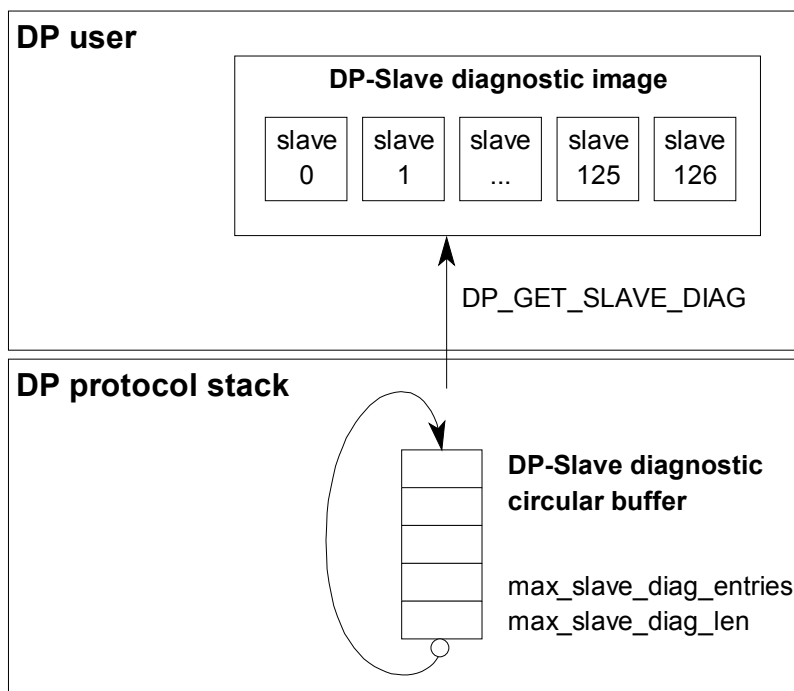


Figure 4-4: DP Slave diagnostic images within the DP and DP User software

The way that the user will be informed about changes within the circular buffer of diagnostic messages depends on the configuration of `cyclic_data_transfer` during the `INIT_MASTER` service.

#### **cyclic\_data\_transfer = TRUE**

The DP Master (class 1) is not synchronised with the DP-User application in this mode and runs independently. If any new diagnostic messages are available the Master will indicate this by means of a `DP_GET_SLAVE_DIAG.ind`. The DP-User application does not have to poll the protocol software. If more than one messages are entered in the circular buffer, the application must retrieve them by means of further `DP_GET_SLAVE_DIAG.req` services.



## cyclic\_data\_transfer = FALSE

After each polling cycle initiated with `DP_DATA_TRANSFER.req` the number of available diagnostic messages within the circular buffer is returned. By means of `DP_GET_SLAVE_DIAG.req` the user can read the oldest diagnostic message from the circular buffer. The number of messages left in the buffer will be also returned.

### Service Description Block for local Request:

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_GET_SLAVE_DIAG
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### Data Block for local Request:

VOID

### Service Description Block for local Confirmation and Indication:

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_GET_SLAVE_DIAG
USIGN8	primitive	CON / IND
INT8	invoke_id	not used
INT16	result	POS/NEG

### Data Block for local Confirmation / Indication:

Data Structure	T_DP_GET_SLAVE_DIAG_CON / T_DP_GET_SLAVE_DIAG_IND	
USIGN16	status	OK, NO
USIGN8	rem_add	0..DP_DEFAULT_STATION_ADDRESS
USIGN8	dummy	alignment byte
INT16	diag_entries	-1..max_diag_entries (see "Init_Master") <sup>9</sup>
USIGN16	diag_data_len	0..DP_MAX_SLAVE_DIAG_DATA_LEN
T_DP_DIAG_DATA	diag_data	

<sup>9</sup> -1 DP\_SLAVE\_DIAG\_OVERFLOW

Data Structure	T_DP_DIAG_DATA	
OCTET	station_status_1	DP_DIAG_1_xxx <sup>10</sup>
OCTET	station_status_2	DP_DIAG_2_xxx <sup>11</sup>
OCTET	station_status_3	DP_DIAG_3_xxx <sup>12</sup>
USIGN8	master_add	Master address which has parameterized the DP Slave
USIGN16	ident_number	PNO ident
OCTET	ext_diag_data [diag_data_len - 6]	

Note: The meaning of the bits in "station\_status\_x" is defined in the standard EN 50170/2 (DP) section "DDLMSlave\_Diag".

<sup>10</sup>	Bit 7	DP_DIAG_1_MASTER_LOCK (influenced by DP Master)
	Bit 6	DP_DIAG_1_PRM_FAULT
	Bit 5	DP_DIAG_1_INVALID_SLAVE_RES (influenced by DP Master)
	Bit 4	DP_DIAG_1_NOT_SUPPORTED
	Bit 3	DP_DIAG_1_EXT_DIAG
	Bit 2	DP_DIAG_1_CFG_FAULT
	Bit 1	DP_DIAG_1_STATION_NOT_READY
	Bit 0	DP_DIAG_1_STATION_NON_EXISTENT (influenced by DP Master)
<sup>11</sup>	Bit 7	DP_DIAG_2_DEACTIVATED (influenced by DP Master)
	Bit 5	DP_DIAG_2_SYNC_MODE
	Bit 4	DP_DIAG_2_FREEZE_MODE
	Bit 3	DP_DIAG_2_WD_ON
	Bit 2	DP_DIAG_2_DEFAULT
	Bit 1	DP_DIAG_2_STAT_DIAG
	Bit 0	DP_DIAG_2_PRM_REQ
<sup>12</sup>	Bit 7	DP_DIAG_3_EXT_DIAG_OVERFLOW

### 4.3.6 Set\_Prm\_Loc

Before starting to communicate with a DP Slave the user has to download the Slave Parameter Set. This includes the following parameter set:

*struct T\_DP\_PRM\_DATA*

USIGN16	prm_data_len	9..DP_MAX_PRM_DATA_LEN
OCTET	station_status	DP_PRM_xxx
OCTET	wd_fact_1	watch dog factors
OCTET	wd_fact_2	t [s] = 10[ms]*wd_fact_1*wd_fact_2
OCTET	min_tsdr	t[bit]
USIGN16	ident_number	PNO ident number
OCTET	group_ident	group member bits
OCTET	prm_user_data [0..DP_MAX_USER_PRM_DATA_LEN]	

The parameter "station\_status" might be changed by the user at any time during the operation phase of a DP Slave (cyclic data exchange).

The service DP\_SET\_PRM\_LOC allows to change the parameter data within the Slave Parameter Set without influencing the Slave status. Thus the operation mode of any DP Slave can be changed also in the data transfer state.

By means of this service the DP\_SL\_NEW\_PRM flag is set and the Slave handler automatically transfers the new parameterization data to the DP Slave.

*Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_SET_PRM_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

*Data Block for local Request:*

Data Structure	T_DP_SET_PRM_REQ	
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
T_DP_PRM_DATA	prm_data	(see above)

*Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_SET_PRM_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

*Data Block for local Confirmation:*

Data Structure	T_DP_SET_PRM_CON	
USIGN16	status	OK, DS, NO

### 4.3.7 Get\_Master\_Diag\_Loc

The service can be used to get the following information:

- current diagnostic status of a specified DP Slave (the data is not taken out of the circular buffer for diagnostic messages!)
- 16 Octets bit field about system diagnostic information (1 bit per DP Slave indicates the availability of new diagnostic messages for the respective Slave)
- Master status information (the operation mode, PNO ident number and release information can be obtained)
- 16 Octets bit field about the data transfer during the "Data\_Control" time (1 bit per DP Slave indicates at least one successful data exchange with the respective Slave)

#### *Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_GET_MASTER_DIAG_LOC
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for local Request:*

Data Structure	T_DP_GET_MASTER_DIAG_REQ	
USIGN8	rem_add	0..125 (address of local Master)
USIGN8	identifier	DP_DIAG_xxx <sup>13</sup>

#### *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_GET_MASTER_DIAG_LOC
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for local Confirmation:*

Data Structure	T_DP_GET_MASTER_DIAG_RES_CON	
USIGN16	status	OK, IV, NO
USIGN16	data_len	0..244
OCTET	diagnostic_data [data_len]	

---

<sup>13</sup>

0..125	DP_DIAG_SLAVE_DATA
126	DP_DIAG_SYSTEM_DIAGNOSTIC
127	DP_DIAG_MASTER_STATUS
128	DP_DIAG_DATA_TRANSFER_LIST

*DP\_DIAG\_SLAVE\_DATA: struct T\_DP\_DIAG\_DATA (see also DP\_GET\_SLAVE\_DIAG)*

OCTET	station_status_1	DP_DIAG_1_xxx
OCTET	station_status_2	DP_DIAG_2_xxx
OCTET	station_status_3	DP_DIAG_3_xxx
USIGN8	master_add	related DP Master to the slave
USIGN16	ident_number	PNO identification
OCTET	ext_diag_data [data_len - DP_MIN_SLAVE_DIAG_LEN]	

*DP\_DIAG\_SYSTEM\_DIAGNOSTIC: OCTET [16]*

OCTET [0]	bit 0 = TRUE: slave 0 has new diag data bit 1 = TRUE: slave 1 has new diag data ...
OCTET [1]	bit 0 = TRUE: slave 8 ...
...	
OCTET [15]	

*DP\_DIAG\_MASTER\_STATUS: struct T\_DP\_MASTER\_STATUS*

OCTET	usif_state	DP_OP_MODE_xxx <sup>14</sup>
USIGN8	ident_number_high	PNO ident number (not aligned)
USIGN8	ident_number_low	
OCTET	dp_hardware_version	DDL/USIF release information
OCTET	dp_firmware_version	
OCTET	user_hardware_version	DP USER release information
OCTET	user_firmware_version	
OCTET	reserved [9]	

*DP\_DIAG\_DATA\_TRANSFER\_LIST: OCTET [16]*

OCTET [0]	bit 0 = TRUE: slave 0 data exchange ok (during Data_Control time) bit 1 = TRUE: slave 1 data exchange ok ...
OCTET [1]	bit 0 = TRUE: slave 8 ...
...	
OCTET [15]	

---

<sup>14</sup>

0x00	DP_OP_MODE_OFFLINE
0x40	DP_OP_MODE_STOP
0x80	DP_OP_MODE_CLEAR
0xC0	DP_OP_MODE_OPERATE

## 4.3.8 Get\_Slave\_Param

This service provides additional local information about Slaves and the DP Master. Additionally all slave diagnostic messages, which are stored in the circular buffer, can be flushed at once. This feature is recommended to use after diagnostic message overrun of the circular buffer.

It can ease the upload of sections from DP Slave parameter sets or to get Master and system wide information. The following data can be obtained in detail:

- all individual parts of any DP Slave parameter set (Header, PRM\_Data, CFG\_Data, AAT\_Data, User\_Data); the upload length is limited up to 244 bytes (DP\_MAX\_TELEGRAM\_LEN)
- Slave specific information (number diag entries, active flag, type, I/O data offsets, I/O length)
- system information (loaded Slaves, active Slaves, number diag entries, I/O image length)

### Service Description Block for local Request:

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_GET_SLAVE_PARAM
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### Data Block for local Request:

Data Structure	T_DP_GET_SLAVE_PARAM_REQ	
USIGN8	identifier	DP_SLAVE_PARAM_xxx <sup>15</sup>
USIGN8	rem_add	0..126 (slave address if necessary)

### Service Description Block for local Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_GET_SLAVE_PARAM
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

### Data Block for local Confirmation:

Data Structure	T_DP_GET_SLAVE_PARAM_CON	
USIGN16	status	OK, NO, IV, NE
USIGN16	data_len	0..244
OCTET	slave_param_data [data_len]	

---

<sup>15</sup>	1	DP_SLAVE_PARAM_HEADER
	2	DP_SLAVE_PARAM_PRM_DATA
	3	DP_SLAVE_PARAM_CFG_DATA
	4	DP_SLAVE_PARAM_AAT_DATA
	5	DP_SLAVE_PARAM_USER_DATA
	6	DP_SLAVE_PARAM_SLAVE_INFO
	7	DP_SLAVE_PARAM_SYS_INFO
	8	DP_SLAVE_PARAM_FLUSH_DIAG

**DP\_SLAVE\_PARAM\_HEADER: struct T\_DP\_SLAVE\_PARAM\_SET**

USIGN16	slave_para_len	22.. MAX_SLAVE_PARAM_LEN (high, low)
USIGN8	sl_flag	DP_SL_ACTIVE, DP_SL_FLAG_NEW_PRM
USIGN8	slave_type	DP_SLAVE_TYPE_DP
OCTET	reserved [12]	

**DP\_SLAVE\_PARAM\_PRM\_DATA: struct T\_DP\_PRM\_DATA**

USIGN16	prm_data_len	2.. DP_MAX_PRM_DATA_LEN (high, low)
OCTET	station_status	DP_PRM_xxx
OCTET	wd_fact_1	watch dog factors
OCTET	wd_fact_2	t [s] = 10[ms]*wd_fact_1*wd_fact_2
OCTET	min_tsd	t[bit]
USIGN16	ident_number	PNO ident number
OCTET	group_ident	group member bits
OCTET	prm_user_data [data_len - DP_MIN_PRM_DATA_LEN]	

**DP\_SLAVE\_PARAM\_CFG\_DATA: struct T\_DP\_CFG\_DATA**

USIGN8	cfg_data_len	2..DP_MAX_CFG_DATA_LEN (high,low)
OCTET	cfg_data [data_len - 2]	see EN 50170/2 (DP) , chapter DDLM_Chk_Cfg

**DP\_SLAVE\_PARAM\_AAT\_DATA: struct T\_DP\_AAT\_DATA**

USIGN16	aat_data_len	2..DP_MAX_AAT_DATA_LEN (high,low)
USIGN8	number_inputs	see "3.6 DPRAM Address Assignment Modes"
USIGN8	number_outputs	
USIGN16	offset_inputs [1..number_inputs]	
USIGN16	offset_outputs [1..number_outputs]	

**DP\_SLAVE\_PARAM\_USER\_DATA: struct T\_DP\_SLAVE\_USER\_DATA**

USIGN16	slave_user_data_len	2..DP_MAX_SLAVE_USER_DATA_LEN (high, low)
OCTET	slave_user_data [slave_user_data_len - 2]	

**DP\_SLAVE\_PARAM\_SLAVE\_INFO: struct T\_DP\_SLAVE\_PARAM\_SLAVE\_INFO**

NOTE: all following values are related to the DP Slave "rem\_add"

USIGN16	diag_entries	number diag messages within the circular buffer
USIGN16	offset_inputs	I/O data location offsets
USIGN16	offset_outputs	
USIGN8	number_inputs	number I/O values in bytes
USIGN8	number_outputs	
USIGN8	sl_flag	DP_SL_ACTIVE, DP_SL_FLAG_NEW_PRM
USIGN8	slave_type	DP_SLAVE_TYPE_DP

**DP\_SLAVE\_PARAM\_SYS\_INFO: struct T\_DP\_SLAVE\_PARAM\_SYS\_INFO**

USIGN8	loaded slaves	number downloaded Slave Parameter Sets
USIGN8	active slaves	number slaves with DP_SL_ACTIVE = TRUE
INT16	diag_entries	number diag entries within the circular buffer <sup>16</sup>
USIGN16	slave_io_image_len	length of used slave I/O memory area

**DP\_SLAVE\_PARAM\_FLUSH\_DIAG:**

VOID	the circular buffer for diagnostic messages has been flushed
------	--------------------------------------------------------------

<sup>16</sup> -1 DP\_SLAVE\_DIAG\_OVERFLOW

### 4.3.7 Set\_Busparameter

This service shall be used if the DP protocol is operated in a multiprotocol environment.

In a multiprotocol environment busparameters are set by means of the FMB service FMB\_Set\_Busparameter but the DP specific section is not contained in that parameter set (e.g. min\_slave\_interval, etc.). The DP parameters are loaded separately with the service DP\_Set\_Busparameters.

#### *Service Description Block for local Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_SET_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for local Request:*

Data Structure	T_DP_SET_BUSPARAMETER_REQ	
USIGN8	bp_flag	bus parameter flags (e.g. DP_BP_ERROR_ACTION)
USIGN8	dummy	alignment byte
USIGN16	min_slave_interval	1..2 <sup>16</sup> -1 [100µs]
USIGN16	poll_timeout	1..2 <sup>16</sup> -1 [1 ms]
USIGN16	data_control_time	1..2 <sup>16</sup> -1 [10 ms]
USIGN16	master_user_data_len	34..DP_MAX_SET_BUSPARAMETER_LEN
STRINGV	master_class2_name [32]	DP Master (class 2) vendor name
OCTET	master_user_data [0..DP_MAX_SET_BUSPARAMETER_LEN - 34]	

#### *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_SET_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for local Confirmation:*

Data Structure	T_DP_SET_BUSPARAMETER_CON	
USIGN16	status	OK, IV, NO



## 4.4 DP MASTER (CLASS 2) SERVICE INTERFACE

Usually a DP Master (class 2) is a device for diagnostic and configuration purposes. That is why services for station address assignment and remote parameterization are provided. The DP Master (class 2) always acts as requester in communication relationships with DP Masters (class 1) and DP Slaves.

The DP Master (class 2) is permitted to use all services offered by the PROFIBUS DP DDLM sublayer. In addition there are some new services that have only been designed to support DP Master (class 2) / DP Master (class 1) communication. The following chapter deals with these services in detail.

### 4.4.1 Upload / Download

These services are used to transfer data blocks between DP Master (class 2) and DP Master (class 1). The DP Master (class 2) always initiates the block transmission.

The data consistency during the transmission of several blocks is not guaranteed by the upload / download services. Additional services for access protection are provided by means of 'Start\_Seq' and 'End\_Seq'. Once 'Start\_Seq' has been called no other user or DP Master (class 2) is allowed to access the destination data until the 'End\_Seq' was sent.

Both services have a local counterpart at Softing's DP Master (class 1) software. Any incoming service will be indicated to the user. The user then may use the local service to handle the indication or rejects the service with a negative service response. (see also chapter "Local / Remote Services" for details)

The upload / download services will be used to transfer all parameter sets necessary for bus communication and Slave handling. Also statistic information can be obtained. Deletion of parameter sets is supported by writing zeros or empty blocks (see also chapter "Upload\_Loc / Download\_Loc" for details).

### Download

#### Service Description Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_DOWNLOAD
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

#### Data Block for Request:

Data Structure	T_DP_DOWNLOAD_REQ	
USIGN16	data_len	0..DP_MAX_DOWNLOAD_DATA_LEN
USIGN8	rem_add	M1 station address
USIGN8	area_code	destination area <sup>17</sup>
USIGN16	add_offset	data block offset
OCTET	data [data_len]	

<sup>17</sup>

0..125	DP_AREA_SLAVE_PARAM (Slave parameter set of the specified DP-Slave)
127	DP_AREA_BUS_PARAM
129	DP_AREA_STAT_COUNT (not implemented yet)
all other values are reserved	

## Data Block for Indication:

Data Structure	T_DP_DOWNLOAD_IND	
USIGN16	data_len	0..DP_MAX_DOWNLOAD_DATA_LEN
USIGN8	rem_add	M2 station address
USIGN8	area_code	see .req
USIGN16	add_offset	data block offset
OCTET	data [data_len]	

## Service Description Block for Response and Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_DOWNLOAD
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS/NEG

## Data Block for Response and Confirmation:

Data Structure	T_DP_DOWNLOAD_RES_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE, TO, FE, NE, AD, EA, LE, SC, NI,
		NC

## Upload

### Service Description Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_UPLOAD
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

### Data Block for Request:

Data Structure	T_DP_UPLOAD_REQ	
USIGN8	rem_add	M1 station address
USIGN8	area_code	destination area (see download)
USIGN16	add_offset	data block starting offset
USIGN8	data_len	1..DP_MAX_UPLOAD_DATA_LEN
USIGN8	dummy	alignment byte

### Data Block for Indication:

Data Structure	T_DP_UPLOAD_IND	
USIGN8	rem_add	M2 station address
USIGN8	area_code	destination area (see download)
USIGN16	add_offset	data block starting offset
USIGN8	data_len	1..DP_MAX_UPLOAD_DATA_LEN
USIGN8	dummy	alignment byte

### Service Description Block for Response and Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_UPLOAD
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS/NEG



*Data Block for Response and Confirmation:*

Data Structure	T_DP_UPLOAD_RES_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE, TO, FE, NE, AD, EA, LE, SC, NI
USIGN16	data_len	0..DP_MAX_UPLOAD_DATA_LEN
OCTET	data [data_len]	

### 4.4.2 Start\_Seq / End\_Seq

These services are intended for use in combination with Upload / Download. They deliver access protection during the block transfer and thus they provide data consistency.

Once the 'Start\_Seq' has been called the user or any other DP Master (class 2) is not permitted to access the DP Master (class 1) as long as the 'End\_Seq' was not called. That means all upload / download sequences may be framed by means of these services and only complete sequences may be accessed.

Both services have a local counterpart at Softing's DP Master (class 1) software. Any incoming service will be indicated to the user. The user then may use the local service to handle the indication or rejects the service with a negative service response (see also chapter "Local / Remote Services for details).

In this implementation the services "Start\_Seq\_Loc / End\_Seq\_Loc" do not provide data access protection. Furthermore they are used to mark the start and the end of download sequences. This mechanism is used to check downloaded data automatically by the protocol stack at the end of the sequence (see chapter "Start\_Seq\_Loc / End\_Seq\_Loc" for details).

#### Start\_Seq

##### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_START_SEQ
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

##### *Data Block for Request:*

Data Structure	T_DP_START_SEQ_REQ	
USIGN8	rem_add	M1 station address
USIGN8	area_code	destination area <sup>18</sup>
USIGN16	timeout	[1ms]

##### *Data Block for Indication:*

Data Structure	T_DP_START_SEQ_IND	
USIGN8	rem_add	M2 station address
USIGN8	area_code	see .req
USIGN16	timeout	[1ms]

##### *Service Description Block for Response and Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_START_SEQ
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS/NEG

<sup>18</sup>

0..125	DP_AREA_SLAVE_PARAM (Slave parameter set of the specified DP-Slave)
127	DP_AREA_BUS_PARAM
129	DP_AREA_STAT_COUNT (not implemented yet)
255	no local access protection
all other values are reserved	

*Data Block for Response and Confirmation:*

Data Structure	T_DP_START_SEQ_RES_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, TO, FE, RE, NE, AD, IP, NI, SE, SC, EA, LE, RE
USIGN8	max_len_data_unit	1..DP_MAX_DOWNLOAD_DATA_LEN
USIGN8	dummy	alignment byte

**End\_Seq**

*Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_END_SEQ
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

*Data Block for Request:*

Data Structure	T_DP_END_SEQ_REQ	
USIGN8	rem_add	M1 station address
USIGN8	dummy	alignment byte

*Data Block for Indication:*

Data Structure	T_DP_END_SEQ_IND	
USIGN8	rem_add	M2 station address
USIGN8	dummy	alignment byte

*Service Description Block for Response and Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_END_SEQ
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS/NEG

*Data Block for Response and Confirmation:*

Data Structure	T_DP_END_SEQ_RES_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, TO, FE, RE, NI, SE, NE, AD, EA, LE, NC

**End\_Seq\_Loc**

*Service Description Block for local Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_END_SEQ_LOC
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	not used

*Data Block for Request:*

Data Structure	T_DP_END_SEQ_LOC_IND	
USIGN16	status	TO

This indication is used to inform about local time-outs during up-/download sequences. The time is based on the time-out value within the "Start\_Seq" service.

### 4.4.3 Act\_Para\_Brct

This service will be sent to one or many DP Slaves to activate a new Bus Parameter Set. The Bus Parameter Set must have been downloaded to all affected stations before!

If a new baud rate is selected all bus stations must be informed in order to switch to the new rate. That is the reason why the service is unconfirmed. Currently only the Bus Parameter Set can be influenced.

All other mode changes to be handled remotely will be affected by means of the 'Act\_Param' service.

In Softing's DP Master (class 1) software the local service "Act\_Param\_Loc" should be used to activate the new Bus Parameter Set when the user receives a `DP_ACT_PARA_BRCT.ind` (see chapter "Local / Remote Services" for details).

#### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_ACT_PARA_BRCT
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for Request and Indication:*

Data Structure	T_DP_ACT_PARA_BRCT_REQ_IND	
USIGN8	rem_add	0..125, DP_GLOBAL_STATION_ADDRESS
USIGN8	area_code	DP_AREA_BUS_PARAM

#### *Service Description Block for local Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_ACT_PARA_BRCT
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for local Confirmation:*

Data Structure	T_DP_ACT_PARA_BRCT_CON	
USIGN16	status	OK, DS

#### 4.4.4 Act\_Param

By means of this service the protocol stack of a DP Master (class 1) can be remotely controlled. The service allows the following settings:

- activation / deactivation of DP Slaves (influences the `DP_SL_ACTIVE` flag in the DP Slave parameter set)
- Bus Parameter Set activation without changing baud rates (use "Act\_Para\_Brct" instead)
- operation mode selection of the DP Master (class 1) (STOP, CLEAR, OPERATE)

In Softing's DP Master (class 1) software the service has to be mapped onto the local counterpart "Act\_Param\_Loc" by the user (see chapter "Local / Remote Services" for details).

##### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP/DP_USR
USIGN8	service	DP_ACT_PARAM
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

##### *Data Block for Request:*

Data Structure	T_DP_ACT_PARAM_REQ	
USIGN8	rem_add	M1 station address
USIGN8	area_code	parameter set to be influenced <sup>19</sup>
USIGN8	activate	area_code dependent value <sup>20</sup>
USIGN8	dummy	alignment byte

##### *Data Block for Indication:*

Data Structure	T_DP_ACT_PARAM_IND	
USIGN8	rem_add	M2 station address
USIGN8	area_code	see .req
USIGN8	activate	see .req
USIGN8	dummy	alignment byte

##### *Service Description Block for Response and Confirmation:*

USIGN16	comm_ref	not used
---------	----------	----------

<sup>19</sup> 0..125 DP\_AREA\_SLAVE\_PARAM (Slave parameter set of the specified DP-Slave)  
 127 DP\_AREA\_BUS\_PARAM no baud rate changes!  
 128 DP\_AREA\_SET\_MODE  
 all other values are reserved

<sup>20</sup> 80H DP\_SLAVE\_ACTIVATE (area\_code = DP\_AREA\_SLAVE\_PARAM)  
 00H DP\_SLAVE\_DEACTIVATE  
 FFH DP\_BUS\_PAR\_ACTIVATE (area\_code = DP\_AREA\_BUS\_PARAM)  
 00H DP\_OP\_MODE\_OFFLINE (area\_code = DP\_AREA\_SET\_MODE)  
 40H DP\_OP\_MODE\_STOP  
 80H DP\_OP\_MODE\_CLEAR  
 C0H DP\_OP\_MODE\_OPERATE





USIGN8	layer	DP/DP_USR
USIGN8	service	DP_ACT_PARAM
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS/NEG

*Data Block for Response and Confirmation:*

Data Structure	T_DP_ACT_PARAM_RES_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, TO, FE, RE, NE, AD, IP, SC, NI, DI, EA, LE

#### 4.4.5 Get\_Master\_Diag

By means of this service the DP Master (class 2) can find out the current status of a specified DP Master (class 1) or get the complete diagnostic data from all assigned DP Slaves. Additionally system-wide and station specific diagnosis can be requested.

##### *Service Description Block for Request and Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_GET_MASTER_DIAG
USIGN8	primitive	REQ/IND
INT8	invoke_id	not used
INT16	result	not used

##### *Data Block for Request:*

Data Structure	T_DP_GET_MASTER_DIAG_REQ	
USIGN8	rem_add	0..125 (DP Master class 1)
USIGN8	identifier	information type <sup>21</sup>

##### *Data Block for Indication:*

Data Structure	T_DP_GET_MASTER_DIAG_IND	
USIGN8	req_add	0..125 (DP Master class 2)
USIGN8	identifier	information type

##### *Service Description Block for Response and Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_GET_MASTER_DIAG
USIGN8	primitive	RES/CON
INT8	invoke_id	not used
INT16	result	POS/NEG

##### *Data Block for Response and Confirmation:*

Data Structure	T_DP_GET_MASTER_DIAG_RES_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE, TO, FE, NE, IP, AD, EA, LE
USIGN16	data_len	0..DP_MAX_MASTER_DIAG_LEN
OCTET	diagnostic data [data_len]	

---

<sup>21</sup>

0..125	DP_DIAG_SLAVE_DATA (Diagnostic data of the specified DP-Slave)
126	DP_DIAG_SYSTEM_DIAGNOSTIC
127	DP_DIAG_MASTER_STATUS
128	DP_DIAG_DATA_TRANSFER_LIST
129..255	reserved

---

## **4.5 DDLM SERVICE INTERFACE**

The PROFIBUS DP Direct Data Link Mapper (DDLM) offers local functions and data transfer functions to be used by the User Interface (USIF) of a DP Master (Class 1) or by the application of a DP Master (Class 2).

In the latter case the DDLM functions are visible at the interface between DP protocol stack and DP application.

In the following the DDLM functions which may be used by a DP Master (Class 2) are described.

The programmer of DP-User application must be aware that some DDLM services may influence the correct operation of the DP Master (class 1)!

### 4.5.1 Set\_Prm

This service is used to transmit parameterization data to a DP Slave. Issuing of that command may be done by a DP Master (class 1) (out of the Slave Parameter Set) or the user of a DP Master (class 2).

It is possible to send parameterization data also in the operation phases of the protocol stack (CLEAR/OPERATE). That means DP Slave operation parameter may be changed at any time (see also "0 Set\_Prm\_Loc" for more information).

#### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_SET_PRM
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for Request:*

Data Structure                      T\_DP\_SET\_PRM\_REQ

USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
T_DP_PRM_DATA	prm_data	

Data Structure                      T\_DP\_PRM\_DATA

USIGN16	prm_data_len	9..DP_MAX_PRM_DATA_LEN (high/low)
OCTET	station_status	DP_PRM_xxx
OCTET	wd_fact_1	watch dog factors
OCTET	wd_fact_2	$t[s] = 10[ms] * wd\_fact\_1 * wd\_fact\_2$
OCTET	min_tsdr	t[bit]
USIGN16	ident_number	PNO ident number
OCTET	group_ident	group member bits
OCTET	prm_user_data	[0..DP_MAX_USER_PRM_DATA_LEN]

#### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_SET_PRM
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for Confirmation:*

Data Structure                      T\_DP\_SET\_PRM\_CON

USIGN16	status	OK, DS, NA, RS, RR, UE, RE
---------	--------	----------------------------

## 4.5.2 Chk\_Cfg

This service has two meanings: First it is used to deliver configuration data to a DP Slave. Second it is intended to be used for configuration checking.

The DP Slave compares the transmitted configuration data with the current data set. If there are inconsistencies between those two sets of data the cyclic data exchange cannot be established. Results of that comparison may be obtained via Slave diagnosis.

### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_CHK_CFG
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### *Data Block for Request:*

Data Structure	T_DP_CHK_CFG_REQ	
USIGN8	rem_add	0..126
USIGN8	dummy	alignment byte
T_DP_CFG_DATA	cfg_data	
Data Structure	T_DP_CFG_DATA	
USIGN16	cfg_data_len	2..DP_MAX_CFG_DATA_LEN (high/low)
OCTET	cfg_data [cfg_data_len - 2]	see EN 50170/2 (DP) , chapter DDLM_Chk_Cfg

### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_CHK_CFG
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

### *Data Block for Confirmation:*

Data Structure	T_DP_CHK_CFG_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE

## 4.5.3 Get\_Cfg

Any DP Master can read the current configuration of a DP Slave. Thus the service is very useful if the current configuration of the DP Slave is unknown. Together with the service "Slave\_Diag" the whole parameter set of any DP Slave can be obtained.

The 'Real\_Cfg\_Data' of the DP Slave will be delivered via the communication system.

### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_GET_CFG
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### *Data Block for Request:*

Data Structure	T_DP_GET_CFG_REQ	
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte

### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_CHK_CFG
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

### *Data Block for Confirmation:*

Data Structure	T_DP_GET_CFG_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
T_DP_CFG_DATA	real_cfg_data	2..DP_MAX_CFG_DATA_LEN

See "Chk\_Cfg" for T\_DP\_CFG\_DATA description.

#### 4.5.4 Slave\_Diag

The current DP Slave status can be read via PROFIBUS by means of this service.

The status information is encoded into the diagnostic data unit. During the startup phase the service call will be repeated cyclically as long as no valid diagnostic data have been received by DP Master (class 1).

##### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_SLAVE_DIAG
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

##### *Data Block for Request:*

Data Structure	T_DP_SLAVE_DIAG_REQ	
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte

##### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_SLAVE_DIAG
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

##### *Data Block for Confirmation:*

Data Structure	T_DP_SLAVE_DIAG_CON	
USIGN16	status	OK, DS, NA, RS, UE, NR, RE
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
USIGN16	diag_data_len	0..DP_MAX_SLAVE_DIAG_DATA_LEN
OCTET	diag_data [diag_data_len]	

If diagnostic data has been available the following structure can be used for the analysis:

Data Structure	T_DP_DIAG_DATA	
OCTET	station_status_1	DP_DIAG_1_xxx <sup>22</sup>
OCTET	station_status_2	DP_DIAG_2_xxx <sup>23</sup>
OCTET	station_status_3	DP_DIAG_3_xxx <sup>24</sup>
USIGN8	master_add	master address which has parameterized the DP Slave
USIGN16	ident_number	PNO ident
OCTET	ext_diag_data [diag_data_len - 6]	

22	Bit 7	DP_DIAG_1_MASTER_LOCK (influenced by DP Master)
	Bit 6	DP_DIAG_1_PRM_FAULT
	Bit 5	DP_DIAG_1_INVALID_SLAVE_RES (influenced by DP Master)
	Bit 4	DP_DIAG_1_NOT_SUPPORTED
	Bit 3	DP_DIAG_1_EXT_DIAG
	Bit 2	DP_DIAG_1_CFG_FAULT
	Bit 1	DP_DIAG_1_STATION_NOT_READY
	Bit 0	DP_DIAG_1_STATION_NON_EXISTENT (influenced by DP Master)
23	Bit 7	DP_DIAG_2_DEACTIVATED (influenced by DP Master)
	Bit 5	DP_DIAG_2_SYNC_MODE
	Bit 4	DP_DIAG_2_FREEZE_MODE
	Bit 3	DP_DIAG_2_WD_ON
	Bit 2	DP_DIAG_2_DEFAULT
	Bit 1	DP_DIAG_2_STAT_DIAG
	Bit 0	DP_DIAG_2_PRM_REQ
24	Bit 7	DP_DIAG_3_EXT_DIAG_OVERFLOW



### 4.5.5 RD\_Inp / RD\_Outp

The user of any DP Master can read a snapshot of the current I/O values of a specified DP Slave.

The DP Slave must already be in the data exchange mode. Use of that service has no influence on the cyclic data exchange with the assigned DP Master (class 1).

The service is intended for diagnostic and configuration purposes only.

#### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_RD_INP / DP_RD_OUTP
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for Request:*

Data Structure	T_DP_RD_INP_REQ	
Data Structure	T_DP_RD_OUTP_REQ	
USIGN8	rem_add	0..126
USIGN8	dummy	alignment byte

#### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_RD_INP / DP_RD_OUTP
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for Confirmation:*

Data Structure	T_DP_RD_INP_CON	
USIGN16	status	OK, DS, NA, RS, UE, NR, RE
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
USIGN16	inp_data_len	0..DP_MAX_INPUT_DATA_LEN
OCTET	inp_data [inp_data_len]	
Data Structure	T_DP_RD_OUTP_CON	
USIGN16	status	OK, DS, NA, RS, UE, NR, RE
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
USIGN16	outp_data_len	0..DP_MAX_OUTPUT_DATA_LEN
OCTET	outp_data [outp_data_len]	

### 4.5.6 Data\_Exchange

This service allows the local DP user to send output data to a DP Slave and receive input data at the same time. The number of transmitted I/O values must have been determined during the configuration procedure.

If diagnostic or error messages are present in the DP Slave a high prior response frame indicates this to the user. The diag flag distinguishes between diagnostic and error messages.

#### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_DATA_EXCHANGE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for Request:*

Data Structure	T_DP_DATA_EXCHANGE_REQ	
USIGN8	rem_add	0..DP_MAX_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
USIGN16	outp_data_len	0..DP_MAX_OUTPUT_DATA_LEN
OCTET	outp_data [outp_data_len]	

#### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_DATA_EXCHANGE
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for Confirmation:*

Data Structure	T_DP_DATA_EXCHANGE_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE
USIGN8	rem_add	0..DP_MAX_SLAVE_ADDRESS
BOOL	diag_flag	TRUE: diagnostic data available
USIGN16	inp_data_len	0..DP_MAX_INPUT_DATA_LEN
OCTET	inp_data [inp_data_len]	

### 4.5.7 Global\_Control

This service allows the DP Master to send control commands to one or many assigned DP Slaves. Grouping of DP Slaves is possible by means of the Group\_Selector bit (multicast service).

The supported control commands depend on the destination station and must have been checked with the "Set\_Prm / Slave\_Diag" services.

#### Service Description Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_GLOBAL_CONTROL
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### Data Block for Request:

Data Structure	T_DP_GLOBAL_CONTROL_REQ	
USIGN8	rem_add	0..126, DP_GLOBAL_STATION_ADDRESS
USIGN8	dummy	alignment byte
USIGN8	control_command	command bits <sup>25</sup> , see also EN 50170/2 (DP)
USIGN8	group_select	see T_DP_PRM_DATA

#### Service Description Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_GLOBAL_CONTROL
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### Data Block for Confirmation:

Data Structure	T_DP_GLOBAL_CONTROL_CON	
USIGN16	status	OK, DS, NO, IV
USIGN8	rem_add	0..126, DP_GLOBAL_STATION_ADDRESS
USIGN8	dummy	alignment byte

<sup>25</sup>

Bit 5	DP_CONTROL_SYNC
Bit 4	DP_CONTROL_UNSYNC
Bit 3	DP_CONTROL_FREEZE
Bit 2	DP_CONTROL_UNFREEZE
Bit 1	DP_CONTROL_CLEAR_DATA
Bits 0, 6, 7	reserved (cleared)

### 4.5.8 Set\_Slave\_Add

This service can be used to modify the station address of a DP Slave.

DP Slaves which do not have any hardware address assignments (switches, EEPROM, etc.) automatically gain the default address (126). Only if the transmitted ident number has matched the stored number the new address will be accepted.

Further address assignment may be forbidden using the "No\_Add\_Chg" flag.

The service can be issued by a DP Master (class 2) only.

#### *Service Description Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DP
USIGN8	service	DP_SET_SLAVE_ADD
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for Request:*

Data Structure	T_DP_SET_SLAVE_ADD_REQ	
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte
USIGN16	rem_slave_data_len	0..DP_MAX_REM_SLAVE_DATA_LEN
USIGN8	new_slave_add	0..DP_MAX_SLAVE_ADDRESS
USIGN8	ident_number_high	PNO ident; alignment problem!
USIGN8	ident_number_low	
BOOL	no_add_chg	TRUE: address change after reset only
OCTET	rem_slave_data [rem_slave_data_len]	

#### *Service Description Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DP_USR
USIGN8	service	DP_SET_SLAVE_ADD
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS/NEG

#### *Data Block for Confirmation:*

Data Structure	T_DP_SET_SLAVE_ADD_CON	
USIGN16	status	OK, DS, NA, RS, RR, UE, RE
USIGN8	rem_add	0..DP_DEFAULT_SLAVE_ADDRESS
USIGN8	dummy	alignment byte

## 5 DATA INTERFACE

The Data Interface is provided by means of the PROFIBUS Application Program Interface (PAPI - see document "User Interface"). It is intended for shared use by host and controller software via DPRAM. It supports fast cyclic I/O data exchange between the user of the DP Master and the assigned DP Slaves.

The layout of the shared memory area is dependent on the chosen Address Assigned Mode. The Data Interface provides the mean to achieve offset and length addressing within a larger memory block.

Two functions handle the DPRAM access with regard to mutual exclusion problems if necessary. They are called from the host using the PAPI functions "profi\_set\_data" and "profi\_get\_data".

The functions have the following prototype:

### profi\_set\_data

*Function prototype:*

```
extern INT16      profi_set_data
(
    IN USIGN8      data_id,
    IN USIGN16     offset,
    IN USIGN16     data_size,
    IN VOID        FAR_D *data_ptr
)
```

*Function parameter description:*

data_id	identifier of the specified memory area
offset	offset within the data structure
data_size	number of bytes to be written to the DPRAM
data_ptr	data block to be written

*Function return values:*

E_OK	(00)	function executed correctly
E_SERVICE_CONSTR_CONFLICT	(23)	service currently not executable, access semaphore has been locked by the controller
E_SERVICE_NOT_SUPPORTED	(24)	identifier is not supported

### profi\_get\_data

*Function prototype:*

```
extern INT16      profi_get_data
(
    IN      USIGN8      data_id,
    IN      USIGN16     offset,
    OUT     USIGN16     FAR_D *data_size,
    INOUT   VOID        FAR_D *data_ptr
)
```

*Function parameter description:*

data_id	identifier of the specified memory area
offset	offset within the data structure
data_size	number of bytes to be read from the DPRAM, returns the number of bytes actually read
data_ptr	data block to be copied to

*Function return values:*

E_OK	(00)	function executed correctly
E_SERVICE_CONSTR_CONFLICT	(23)	service currently not executable, access semaphore has been locked by the controller
E_SERVICE_NOT_SUPPORTED	(24)	identifier is not supported

**Shared memory access without DPRAM**

For applications without DPRAM the protocol stack automatically provides the DP Slave data and status areas out of the dynamic memory pool. Accessing these areas may be done by means of the following as "extern" defined pointers:

- `slave_image_ptr, slave_image_len`: pointer and length of the DP Slave I/O data area
- `slave_status_ptr, DP_STATUS_INFO_LEN`: pointer and length of the DP status information field;  
in cases the status is not supported the "slave\_status\_ptr" will be NULL

The user must take care on data consistency by himself. If the service "Data\_Transfer" is used and no data access by the user occurs during the service execution then data consistency is gained automatically.

## 5.1 DP SLAVE I/O DATA ACCESS

To access the DP Slave input / output data the functions "profi\_set\_data" and "profi\_get\_data" can be used. The location of the input / output bytes of a specified DP Slave depends on the DPRAM Address Assignment Mode (AAM).

To calculate the correct offset and address the respective I/O bytes the following formulas may be used (for the parameter description see service "FMB\_Set\_Configuration"):

### ARRAY - Address Assignment Mode

```
in_offset  = (slave_address - lowest_slave_address) * max_slave_input_len;
out_offset = (slave_address - lowest_slave_address) * max_slave_output_len +
             offset_inputs;
```

### USER DEFINED - Address Assignment Mode

```
T_DP_AAT_DATA  *aat_data_ptr = ...; /* see Slave Parameter Set */
in_offset  [i] = aat_data_ptr->offset_input [i];
out_offset [i] = aat_data_ptr->offset_output [i];
```

### COMPACT - Address Assignment Mode

```
T_DP_AAT_DATA *aat_data_ptr = ...; /* see Slave Parameter Set */
in_offset  = aat_data_ptr->offset_input [0];
out_offset = aat_data_ptr->offset_output [0];
```

### IO-BLOCK - Address Assignment Mode

see chapter 3.6.4

## 5.2 STATUS INFORMATION

The Status Information array is provided to report the current operating state of the DP communication participants to the user. It describes always the state of all activated stations (DP Slaves, DP Master) during the **last polling cycle**.

The Status Information is optional and can be enabled/disabled via compile time variables. If Status Information is enabled the following features can be used:

## DP protocol stack ⇒ DP user

One status byte per communication station will be updated cyclically by the DP Master (class 1) before finishing the polling cycle. It delivers an image about the error state of any activated Slave and the existence of diagnostic messages. The user may evaluate the information to decide whether diagnostic or process data has to be read from a specified DP Slave.

The end of DP Master polling cycles can only be recognized by means of a `DATA_TRANSFER.con` in the initialisation mode `cyclic_data_transfer = FALSE` (see service `INIT_MASTER`).

The following figure depicts the structure of the status information array:

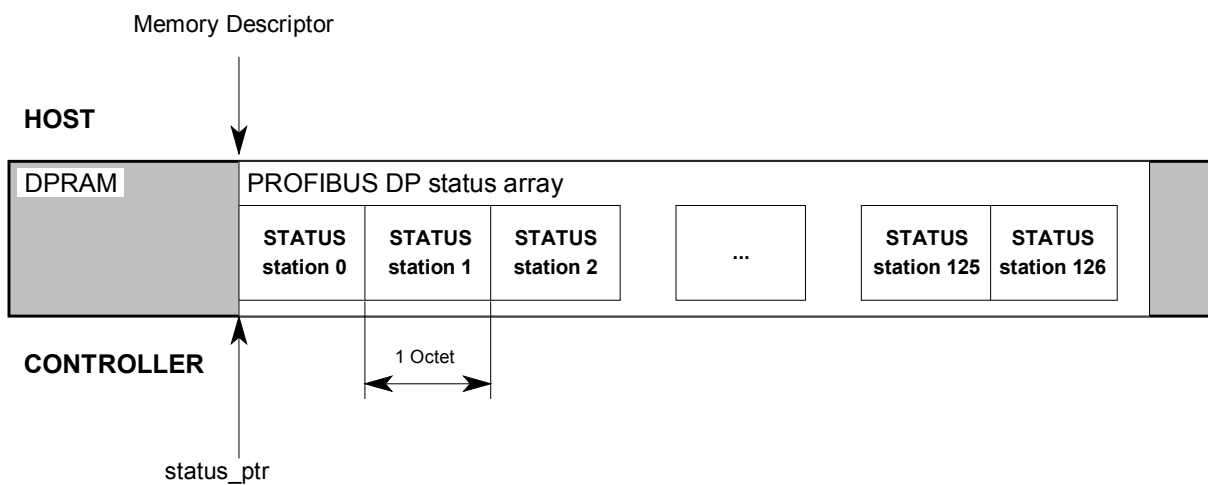


Figure 5-1: DPRAM status information array

## Coding of the Status Information Bits

### Station type identifier:

Bit 7	DP_STATE_STATION_TYPE	1: DP_STATE_MASTER_STATION
		0: DP_STATE_SLAVE_STATION

### DP Slave Status Octet:

Bit 6..2	reserved (cleared)	
Bit 1	DP_STATE_SLAVE_DIAG_DATA	1: new diagnostic data available
Bit 0	DP_STATE_SLAVE_ERROR	1: no valid communication to the slave

### DP Master (class 1) Status Octet:

Bit 6..2	reserved (cleared)	
Bit 1	DP_STATE_MASTER_DIAG_DATA	1: diagnostic data transmission to the DP Master (class 2) is in process
Bit 0	DP_STATE_MASTER_UP_DOWN_LOAD	1: upload / download service is in process



## 6. DP STATUS AND ERROR CODES

### 6.1. CODING CONVENTIONS

Each DP service confirmation returns an USIGN16 status value which is defined as follows:

- **USIGN8 (low byte):** Contains the error code. This byte indicates general errors like "Invalid Parameter", "Not Implemented", etc.
- **USIGN8 (high byte):** This byte is used only if an error extension is available. By means of this byte additional error or status information are reported to the user.

**Example:** Error Code = 0x03C7 ⇒ E\_DP\_WRONG\_SLAVE\_ADD and E\_DP\_IP

### 6.2. ERROR CODE DEFINITIONS

#### 6.2.1. Error Codes

E_DP_OK = 0x00:	OK, acknowledgement positive
E_DP_UE = 0x01:	remote user error (the reason is not visible for this station)
E_DP_RR = 0x02:	resources at remote station are insufficient (e.g. no memory at service access point (SAP) available at this time)
E_DP_RS = 0x03	service or service access point (SAP) at remote station is deactivated
E_DP_RA = 0x04	access of remote service access point (SAP) has been blocked
E_DP_NA = 0x11	no reaction from remote station, i.e. the station physically does not respond (check power supply, cabling or the state of that remote station)
E_DP_DS = 0x12	local entity disconnected, i.e. this station was not able to send (check the bus connection of this station)
E_DP_NO = 0x13	not possible in this state, i.e. the requested service cannot be executed in this state of the station or is not possible under these circumstances (see error extension)
E_DP_LR = 0x14	local resource not available (internal FDL error)
E_DP_IV = 0x15	invalid parameter in request, i.e. the service request contains at least one syntactically wrong parameter (e.g. range limits exceeded, etc.)
E_DP_TO = 0x16	service timeout expired
E_DP_FE = 0xC1	format error in request frame, e.g. during Master / Master communication an invalid request has been received from the remote master

E_DP_NI = 0xC2	the requested function is not implemented
E_DP_AD = 0xC3	access denied
E_DP_EA = 0xC4	area too large, e.g. the used data block is too large to fit in the memory of the remote station
E_DP_LE = 0xC5	data block length exceeded, i.e. the data block length exceeds the provided area
E_DP_RE = 0xC6	format error in response frame, e.g. during Master / Master communication an invalid response has been received from the remote master
E_DP_IP = 0xC7	invalid parameter, i.e. the service request contains at least one unknown or wrong parameter (e.g. slave not available because the slave has not been downloaded)
E_DP_SC = 0xC8	sequence conflict, i.e. the service is not executable at this time and operation mode
E_DP_SE = 0xC9	sequence error, i.e. the service was used in the wrong context (e.g. DP_END_SEQ before DP_START_SEQ)
E_DP_NE = 0xCA	area non-existent, e.g. a non-existent slave parameter set cannot be uploaded, etc.
E_DP_DI = 0xCB	data incomplete
E_DP_NC = 0xCC	the used master parameter set is incompatible

### 6.2.2. Error Code Extensions

E_DP_DATA_ALIGNMENT = 0x01	<ul style="list-style-type: none"><li>- the slave parameter set is inconsistent (e.g. wrong length - check slave_para_len, cfg_len, prm_len, add_tab_len, user_len and the alignment of the format)</li><li>- the starting offset of input or output data is odd and must be aligned to an even address</li></ul>
E_DP_TOO_MANY_SLAVES = 0x02	<ul style="list-style-type: none"><li>- the number of slave parameter sets exceeds Max_Number_Slaves (see FMB_Set_Configuration)</li></ul>
E_DP_WRONG_SLAVE_ADD = 0x03	<ul style="list-style-type: none"><li>- the used slave address is out of range (0..125, 126)</li><li>- the slave address is identical with this station</li><li>- DP_AAM_ARRAY: the slave address is lower than Lowest_Slave_Address (see DP_Init_Master)</li><li>- DP_AAM_ARRAY: the slave address is higher than Lowest_Slave_Address + Max_Number_Slaves - 1</li></ul>
E_DP_AAM_NOT_SUPPORTED = 0x04	<ul style="list-style-type: none"><li>- the Address Assignment Mode "User Defined" cannot be used with this hard- or firmware release</li></ul>
E_DP_TOO_FEW_DIAG_ENTRIES = 0x05	<ul style="list-style-type: none"><li>- not used</li></ul>

E_DP_WRONG_PRM_DATA_LEN = 0x06	<ul style="list-style-type: none"> <li>- the parameter data set by means of service DP_Set_Prm_Loc has a different length than used during DP_Download_Loc</li> <li>- the parameter data length range was exceeded (7..244)</li> </ul>
E_DP_WRONG_CFG_DATA_LEN = 0x07	<ul style="list-style-type: none"> <li>- the config data length range was exceeded (1..244)</li> </ul>
E_DP_WRONG_DIAG_LEN = 0x08	<ul style="list-style-type: none"> <li>- not used</li> </ul>
E_DP_WRONG_BUS_PARA_LEN = 0x09	<ul style="list-style-type: none"> <li>- the bus parameter set does not fit the configured length (see service FMB_Set_Configuration - Max_Bus_Para_Len)</li> <li>- the used add_offset exceeds the length of the bus parameter set</li> </ul>
E_DP_WRONG_SLAVE_PARA_LEN = 0x0A	<ul style="list-style-type: none"> <li>- the slave parameter set does not fit the configured length (see service FMB_Set_Configuration - Max_Slave_Para_Len)</li> <li>- the used add_offset exceeds the length of the bus parameter set</li> </ul>
E_DP_WRONG_IO_DATA_LEN = 0x0B	<ul style="list-style-type: none"> <li>- the number of inputs or outputs defined in the slave parameter set exceeds the configured length (see service FMB_Set_Configuration - Max_Slave_Input_Len, Max_Slave_Output_Len)</li> <li>- the number of inputs or outputs defined for that slave are longer than allowed (0..244)</li> </ul>
E_DP_NOT_ENOUGH_MEMORY = 0x0C	<ul style="list-style-type: none"> <li>- not used</li> </ul>
E_DP_WRONG_USIF_STATE = 0x0D	<ul style="list-style-type: none"> <li>- the master is not able to execute the requested service in the current state:</li> <li>- FMB_Set_Configuration and / or DP_Init_Master were not called (still OFFLINE operation mode)</li> <li>- DP_Act_Param_Loc can only change to the next operation mode (e.g. STOP to CLEAR, not STOP to OPERATE)</li> </ul>
E_DP_SLAVE_ACCESS_DENIED = 0x0E	<ul style="list-style-type: none"> <li>- not used</li> </ul>
E_DP_WRONG_AREA_CODE = 0x0F	<ul style="list-style-type: none"> <li>- during a download or upload sequence different area_codes were detected</li> <li>- the used area_code is out of range (0..0x81, 0xFF)</li> </ul>
E_DP_NOT_SUPPORTED = 0x10	<ul style="list-style-type: none"> <li>- not used</li> </ul>
E_DP_PRM_DATA_FAULT = 0x11	<ul style="list-style-type: none"> <li>- the slave watchdog factor values are zero and the watchdog has been enabled</li> </ul>
E_DP_CFG_DATA_FAULT = 0x12	<ul style="list-style-type: none"> <li>- not used</li> </ul>

E_DP_AAT_DATA_FAULT = 0x13	<ul style="list-style-type: none"> <li>- DP_AAM_ARRAY: aat_data_len is invalid (&lt; 2)</li> <li>- DP_AAM_COMPACT: the number of inputs or outputs within the Address Assignment Table is inconsistent to the config data</li> <li>- DP_AAM_COMPACT: aat_data_len is invalid (6 or 8)</li> <li>- DP_AAM_DEFINED: the number of inputs or outputs within the Address Assignment Table is inconsistent to the config data</li> <li>- DP_AAM_DEFINED: aat_data_len is invalid (<math>4 + 2 * \text{number\_inputs} + 2 * \text{number\_outputs}</math>)</li> </ul>
E_DP_USER_DATA_FAULT = 0x14	<ul style="list-style-type: none"> <li>- not used</li> </ul>
E_DP_SLAVE_PARA_FAULT = 0x15	<ul style="list-style-type: none"> <li>- an invalid sl_flag within the slave parameter set was detected (0x80: Active, 0x40: New_Prm, 0x20: Fail_Safe)</li> </ul>
E_DP_AREA_NOT_ACCESSED = 0x16	<ul style="list-style-type: none"> <li>- the service DP_End_Seq was received but no upload or download has occurred</li> </ul>
E_DP_WRONG_BAUDRATE = 0x17	<ul style="list-style-type: none"> <li>- an illegal baudrate constant was used (use DP_KBAUD_xxx)</li> </ul>
E_DP_WRONG_BP_FLAG = 0x18	<ul style="list-style-type: none"> <li>- an invalid bp_flag was detected (0x80: DP_BP_ERROR_ACTION - Autoclear)</li> </ul>
E_DP_WRONG_FDL_STATE = 0x19	<ul style="list-style-type: none"> <li>- not used</li> </ul>
E_DP_WRONG_ACTIVATION = 0x1A	<ul style="list-style-type: none"> <li>- an invalid (de)activation constant was used (0x80: DP_SLAVE_ACTIVATE, 0x00: DP_SLAVE_DEACTIVATE, 0xFF: DP_BUS_PAR_ACTIVATE, 0x00: DP_OP_MODE_OFFLINE, 0x40: DP_OP_MODE_STOP, 0x80: DP_OP_MODE_CLEAR, 0xC0: DP_OP_MODE_OPERATE)</li> </ul>
E_DP_WRONG_MASTER_ADD = 0x1B	<ul style="list-style-type: none"> <li>- the master address is out of range (0..125)</li> <li>- the remote master address is identical with the local master address</li> </ul>
E_DP_DPRAM_INIT_ERROR = 0x1C	<ul style="list-style-type: none"> <li>- not used (FMB_Set_Configuration.con [-] instead)</li> </ul>
E_DP_WRONG_LEN = 0x1D	<ul style="list-style-type: none"> <li>- the upload or download data length is larger than allowed (0..240)</li> </ul>
E_DP_WRONG_IDENTIFIER = 0x1F	<ul style="list-style-type: none"> <li>- the identifier of service DP_Get_Master_Diag is out of range (0..0x80)</li> <li>- the identifier of service DP_Get_Slave_Param is wrong or not supported by your firmware version</li> </ul>
E_DP_LOAD_BUS_PARAMETER = 0x20	<ul style="list-style-type: none"> <li>- the attempt to set new bus parameters by DP failed (illegal values within bus parameters or wrong state)</li> </ul>

- E\_DP\_ACTIVATE\_SAP = 0x21 - DP was not able to activate the necessary service access points (SAPs); another application might work corrupt
- E\_DP\_WRONG\_REMOTE\_SERVICE = 0x22 - service DP\_Init\_Master detected an invalid value in parameter auto\_remote\_services:  
(0x80: DP\_AUTO\_GET\_MASTER\_DIAG,  
0x40: DP\_AUTO\_UPLOAD\_DOWNLOAD\_SEQ,  
0x20: DP\_AUTO\_ACT\_PARAM)



# **PROFIBUS Application Program Interface**

## **DP/V1 Services**

Version 5.2  
Rev. 00

Date: 17-October-1997

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---



**CONTENTS**

1 SCOPE .....1

2 OVERVIEW .....2

3 DP/V1 INITIATE AND ABORT SERVICES .....4

    3.1 DP\_INITIATE.....4

    3.2 DP\_ABORT .....8

4 DP/V1 READ AND WRITE SERVICES.....10

    4.1 DP\_READ .....10

    4.2 DP\_WRITE.....12

5 NEGATIVE CONFIRMATIONS, ERROR- AND RETURN CODES .....14

    5.1 NEGATIVE CONFIRMATIONS.....14

    5.2 RETURN CODES .....16



## 1 SCOPE

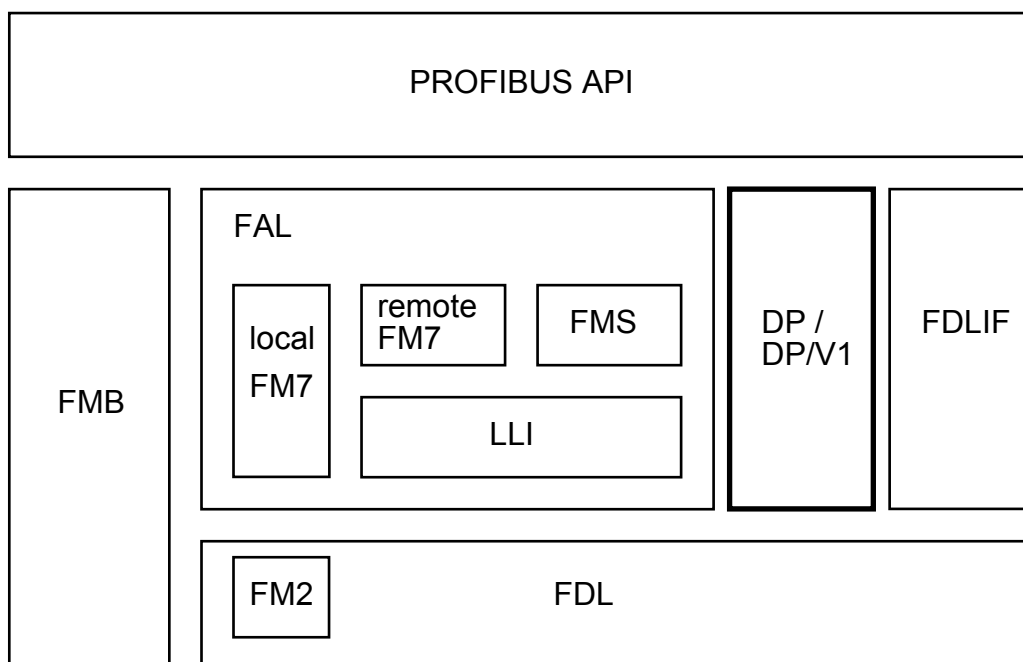
This manual describes the programming interface between PROFIBUS DP/V1 protocol software and PROFIBUS DP/V1 application.

This manual does not describe the functionality of PROFIBUS DP/V1. Therefore, it is expected that the reader is familiar with PROFIBUS DP/V1 and that he knows EN 50170/2.

Softing's PROFIBUS Application Program Interface provides uniform access to all service groups of the PROFIBUS protocol. The common access functions are described in the "User Interface" part of the PROFIBUS User Manual.

This manual describes the service-specific parameters and data for DP/V1 Master-Slave acyclic services Class 2 (MSAC\_C2) specified in PROFIBUS Draft Specification V1.13 (March 7, 1997).

The DP/V1 specific types and constants are defined in the include file PB\_DP.H.



This document should be read in conjunction with the following parts of the PROFIBUS User Manual:

- "User Interface" (describes the uniform access functions to all PROFIBUS services)
- "Basic Management" (describes the management services common to all protocol components)
- "DP Services" (describes the DP services)

## 2 OVERVIEW

DP/V1 provides services, which refers to MSAC\_C2 Client functionality.

**Simultaneous operation of DP/V1 and FMS is not possible. In this case all DP/V1 services will be rejected.** The operation mode for DP/V1 has to be set with the FMB\_SET\_CONFIGURATION service with the parameters FMS\_ACTIVE = PB\_FALSE and DP\_ACTIVE = PB\_TRUE (see manual Basic Management chapter 3.1).

The service descriptions in this manual are grouped in Context Management Services and Data Transfer-Services:

**Context Management** services are used to establish a connection and to release a connection.

**Data Transfer** services are used to read data from a slave and write data to a slave.

The following table shows the services and their appearance as Requests, Confirmations or Indications.

	MSAC_C2 Request	MSAC_C2 Confirmation	MSAC_C2 Indication
DP_INITIATE	X	X	
DP_READ	X	X	
DP_WRITE	X	X	
DP_ABORT	X		X

Before using DP/V1 services, the DP protocol stack has to be in the state *stop*, *clear* or *operate*, otherwise there would be no access to the physical layer (see manual DP Services chapter 3.2).

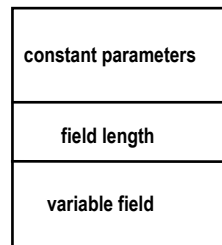
## Notes on Data Structures and Parameters

The DP/V1 specific types and constants are defined in the include file PB\_DP.H.

All words, long-words, strings, arrays and records begin on even addresses. To accomplish this, fill bytes had to be added in some places. These are always recognizable by their name *dummy*.

Data blocks do not contain pointers. If a data block contains one or more fields or lists of variable length, then the length information of all variable-length fields is stored in the constant part. The fields of variable length follow on the constant part.

Here is an example of such a data block:



The variable data fields are shown between comment delimiters in the include file PB\_DP.H to show their position and structure, without forcing the programmer to use data structures of a specific length. Nevertheless, the data must be entered at exactly this spot.

The service description block contains a *result* parameter. If a function returns as positive (result = POS) the service-specific confirmation block will be passed. If the result is negative (result = NEG), then the error structure T\_DP\_ERROR\_CON is passed.

For negative confirmations a standard error structure T\_DP\_ERROR\_CON is used. The error codes are not noted explicitly for each service. The standard error structure and the error codes are described in chapter 5 "Negative Confirmations, Error- and Return Codes".

## 3 DP/V1 INITIATE AND ABORT SERVICES

The DP-Master (Class 2) uses the INITIATE- and ABORT services to establish or release a communication relationship to the DP-Slave.

### 3.1 DP\_INITIATE

DP\_INITIATE establishes a connection to a slave.

Each connection is identified by its communication reference (comm\_ref). The service DP\_INITIATE connects a comm\_ref with a slave's address.

The implementation supports several (DP\_MAX\_CHANNELS\_MSAC2=30) parallel connections. On each connection, there is only one service allowed at one time, except the DP\_ABORT service, which is allowed at any time.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP
USIGN8	service	DP_INITIATE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

Data structure	T_DP_INITIATE_REQ		
USIGN8	rem_add	remote station address	(0..126)
USIGN8	reserved[3]	reserved for future use	
USIGN16	send_timeout	control time for supervision in 10 ms units	(0..65535)
OCTET	features_supported[2]	supported features of the master	{0x01, 0x00}
OCTET	profile_features_supported[2]	supported features regarding used profiles	{0x00, 0x00}
USIGN16	profile_ident_number	profile ident	
T_ADD_ADDR	add_addr_param	extended address scheme	

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP_USR
USIGN8	service	DP_INITIATE
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS

*Data block for Confirmation:*

Data structure	T_DP_INITIATE_CON		
USIGN16	status	status	(E_DP_OK)
USIGN8	rem_add	remote station address	(0..126)
USIGN8	max_len_data_unit	maximum data-length, the slave can provide	(1..240)
OCTET	features_supported[2]	supported features of the slave	{0x01, 0x00}
OCTET	profile_features_supported[2]	supported features regarding used profiles	{0x00, 0x00}
USIGN16	profile_ident_number	profile ident number	
T_ADD_ADDR	add_addr_param	extended address scheme	

## send\_timeout and max\_len\_data\_unit

- *send\_timeout*, the base time interval, used by master and slave for initiating Idle-Sequences, to find out, whether the Remote-Partner is still in operation
- *max\_len\_data\_unit*, the maximum Data-Length, the slave supports.

## Extended Address Scheme

The DP\_INITIATE service offers the possibility to communicate over multiple interconnected networks.

- Data Transfer from one DP-Network to another DP-Network over a link
- Data Transfer from an alien network to DP-Network over a gateway

### Data Structure T\_ADD\_ADDR

USIGN8	s_type	indicates the presence of the optional network source address
USIGN8	s_len	indicates the length of the s_addr parameter
USIGN8	d_type	indicates the presence of the optional network destination
address		
USIGN8	d_len	indicates the length of the d_addr parameter
T_ADDR	s_addr	additional source address information
T_ADDR	d_addr	additional destination address information

### Data Structure T\_ADDR

USIGN8	api	application process instance
USIGN8	scl	security level
OCTET	network_address[6]	identifies the network address
OCTET	mac_address[length-8]	identifies the MAC address

If the extended address scheme is not used (default case), the parameter T\_ADDR\_ADDR add\_addr\_param in the DP\_INITIATE-Request service has the following content:

## Data Structure T\_ADD\_ADDR

USIGN8	s_type	0
USIGN8	s_len	2
USIGN8	d_type	0
USIGN8	d_len	2
T_ADDR	s_addr	see below
T_ADDR	d_addr	see below

## Data Structure T\_ADDR

USIGN8	api	0..255, usually 0
USIGN8	scl	0..255, usually 0

If the extended address scheme is used (extended case), the parameter T\_ADDR\_ADDR add\_addr\_param in the DP\_INITIATE-Request service has the following content:

## Data Structure T\_ADD\_ADDR

USIGN8	s_type	0 or 1
USIGN8	s_len	s_type = 0, length of s_addr is 2 s_type = 1, length of s_addr is at least 8
USIGN8	d_type	0 or 1
USIGN8	d_len	d_type = 0, length of d_addr is 2 d_type = 1, length of d_addr is at least 8
T_ADDR	s_addr	see below
T_ADDR	d_addr	see below

## Data Structure T\_ADDR

USIGN8	api	0..255, usually 0
USIGN8	scl	0..255, usually 0
OCTET	network_address[6]	use only if [d]s_type = 1
OCTET	mac_address[[d]s_len-8]	use only if [d]s_type = 1

There are two macros (defined in PB\_DP.H) to calculate the extended addresses s\_addr and d\_addr.

- DP\_INITIATE\_S\_ADDR requires a pointer to a T\_DP\_INITIATE\_REQ or a T\_DP\_INITIATE\_CON and gives a pointer of type T\_ADDR to the s\_addr field.
- DP\_INITIATE\_D\_ADDR requires a pointer to a T\_DP\_INITIATE\_REQ or a T\_DP\_INITIATE\_CON and gives a pointer of type T\_ADDR to the d\_addr field, provided that the entry s\_len in the add\_addr\_param field is set correctly.

According to PROFIBUS Draft Specification V1.13 (March 7,1997), the slave has to swap the s\_addr and the d\_addr field in the response, the master simply passes those parameters without further checking.



## Technical Details

Initializing a connection requires some communication between master and slave. First, the master sends the DP\_INITIATE request to the slave's resource manager. The slave sends a Resource-Management request to the master and tells the master, whether there is a free SAP left on the slave and which one to use. If there is no SAP free (because too many connections have already been opened), the user will receive an DP\_ABORT indication with abort reason DP\_ABORT\_FDL\_RS. This error may occur after aborting a connection to a slave and sending a DP\_INITIATE request immediately afterwards. In this case the error will be only temporary and will disappear when the slave has terminated its internal state-machines. If the slave does not support DP/V1 this DP\_ABORT indication will occur, too.

If there is a free SAP but the slave does not support the send-timeout interval requested by the master's user, there also will be an DP\_ABORT indication with abort reason DP\_ABORT\_DDLM\_ABT\_STO. The additional detail contains the minimum send\_timeout interval that the slave supports actually. The master can choose any greater send\_timeout interval than the slave's minimum send\_timeout interval (see also chapter 3.2 DP\_ABORT service).

Having received the Resource-Management request, the master polls the received SAP and gets a DP\_INITIATE confirmation, which will be passed to the user. The user should keep the max\_len\_data\_unit, as it limits the length of the data-buffers, which can be passed between master and slave. Both master and slave will check the data length of the DP\_READ-/ DP\_WRITE requests which follow the DP\_INITIATE.

## 3.2 DP\_ABORT

With the DP\_ABORT service, an open connection to a slave may be released. The connection may be released by the master or by the slave. The DP\_ABORT service is also called by the master's protocol software, whenever communication errors are detected.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP / DP_USR
USIGN8	service	DP_ABORT
USIGN8	primitive	REQ / IND
INT8	invoke_id	not used / 0
INT16	result	not used

### *Data block for Request:*

Data structure	T_DP_ABORT_REQ		
USIGN8	subnet	net location	(NO, SUBNET_LOCAL,
SUBNET_REMOTE)			
USIGN8	reason	reason	(0..0x3F)

### *Data block for Indication:*

Data structure	T_DP_ABORT_IND		
USIGN8	locally_generated	PB_FALSE, PB_TRUE	
USIGN8	subnet	net location	(NO, SUBNET_LOCAL,
SUBNET_REMOTE)			
USIGN8	reason	instance / reason	(0..0xFF)
USIGN8	dummy	alignment byte	
USIGN16	additional_detail	additional detail	(see below)

Using a gateway in the network, the parameter SUBNET indicates the location of the abort in the network.

Subnet:

NO	0	location between Master and Gateway or Slave
SUBNET_LOCAL	1	location between the Gateway and Slave
SUBNET_REMOTE	2	location between Slave and Gateway

The parameter REASON contains the abort protocol instance and the abort reason.

- The upper two bits represent the protocol instance:

DP_ABORT_INSTANCE_FDL	0x00	protocol instance FDL
DP_ABORT_INSTANCE_DDLM	0x40	protocol instance DP
DP_ABORT_INSTANCE_USER	0x80	protocol instance USER

- The lower six bits represent the reason:

Remote abort reasons occurred in the protocol instance DP\_ABORT\_INSTANCE\_FDL:

DP_ABORT_FDL_UE	(DP_ABORT_INSTANCE_FDL   0x01)
DP_ABORT_FDL_RR	(DP_ABORT_INSTANCE_FDL   0x02)
DP_ABORT_FDL_RS	(DP_ABORT_INSTANCE_FDL   0x03)
DP_ABORT_FDL_NR	(DP_ABORT_INSTANCE_FDL   0x09)
DP_ABORT_FDL_DH	(DP_ABORT_INSTANCE_FDL   0x0A)
DP_ABORT_FDL_RDL	(DP_ABORT_INSTANCE_FDL   0x0C)
DP_ABORT_FDL_RDH	(DP_ABORT_INSTANCE_FDL   0x0D)

Additional local abort reasons occurred in the protocol instance DP\_ABORT\_INSTANCE\_FDL:

DP_ABORT_FDL_LS	(DP_ABORT_INSTANCE_FDL   0x10)
DP_ABORT_FDL_NA	(DP_ABORT_INSTANCE_FDL   0x11)
DP_ABORT_FDL_DS	(DP_ABORT_INSTANCE_FDL   0x12)
DP_ABORT_FDL_NO	(DP_ABORT_INSTANCE_FDL   0x13)
DP_ABORT_FDL_LR	(DP_ABORT_INSTANCE_FDL   0x14)
DP_ABORT_FDL_IV	(DP_ABORT_INSTANCE_FDL   0x15)

Abort reasons occurred in the protocol instance DP\_ABORT\_INSTANCE\_DDLM:

DP_ABORT_DDLM_ABT_SE	(DP_ABORT_INSTANCE_DDLM   0x01)
DP_ABORT_DDLM_ABT_FE	(DP_ABORT_INSTANCE_DDLM   0x02)
DP_ABORT_DDLM_ABT_TO	(DP_ABORT_INSTANCE_DDLM   0x03)
DP_ABORT_DDLM_ABT_RE	(DP_ABORT_INSTANCE_DDLM   0x04)
DP_ABORT_DDLM_ABT_IV	(DP_ABORT_INSTANCE_DDLM   0x05)
DP_ABORT_DDLM_ABT_STO	(DP_ABORT_INSTANCE_DDLM   0x06)
DP_ABORT_DDLM_ABT_IA	(DP_ABORT_INSTANCE_DDLM   0x07)
DP_ABORT_DDLM_ABT_OC	(DP_ABORT_INSTANCE_DDLM   0x08)

Abort reasons occurred in the instance DP\_ABORT\_INSTANCE\_USER:

The PROFIBUS Draft Specification V1.13 (March 7, 1997) specifies **no** user abort reasons.

Additional Detail:

An additional detail exists for the abort reason DP\_ABORT\_DDLM\_ABT\_STO. In this case this parameter contains the minimal **send\_timeout** (1..65535) interval supported by the slave.

**Hint:** On the bus, instance and reason are encoded slightly different. The lower four bits represent the reason, the next two bits represent the instance and the two most significant bits are reserved. Keep that in mind when watching the bus traffic with a monitoring device.

## 4 DP/V1 READ AND WRITE SERVICES

### 4.1 DP\_READ

The DP\_READ service reads data from an object identified by slot\_number and index of a DP Slave. Usually the slot\_number refers to a physical device in a modular slave and the index identifies an object within this module.

The confirmation of this service contains the read data.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP
USIGN8	service	DP_READ
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

Data structure	T_DP_READ_REQ	
USIGN8	rem_add	remote address (not evaluated on MSAC_C2 connections)
USIGN8	slot_number	slot number (0..254)
USIGN8	index	index (0..254)
USIGN8	length	data length (0..240) , restricted by the results of the initiate-confirmation

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP_USR
USIGN8	service	DP_DP_READ
USIGN8	primitive	CON
INT8	invoke_id	unchanged
INT16	result	POS / NEG

## Data block for Confirmation:

result = POS:

Data structure	T_DP_READ_CON	
USIGN16	status	status (E_DP_OK)
USIGN8	rem_add	remote address (0..126)
USIGN8	slot_number	slot number (0..254)
USIGN8	index	index (0..254)
USIGN8	length	data length (requested data length)
USIGN8	data[length]	read data

result = NEG

Data structure	T_DP_ERROR_CON	standard DP error structure (see chapter 5 about Error Structure and Error Codes)
----------------	----------------	--------------------------------------------------------------------------------------

## NOTES:

- The requested number of bytes should at least be equal to the number of bytes the slave's object will deliver. Otherwise, the slave might refuse to transfer any data (negative confirmation with status E\_DP\_UE).
- While the master is waiting for the response, no further requests can be issued and will be rejected with interface error E\_IF\_NO\_PARALLEL\_SERVICES, except the DP\_ABORT service.
- If the connection is closed, the service will be rejected with interface error E\_IF\_SERVICE\_NOT\_EXECUTABLE.
- The rem\_add, slot\_number and index in the confirmation are redundant informations (mirroring the request's data) and need not be evaluated.

## 4.2 DP\_WRITE

The DP\_WRITE service writes data to an object identified by slot\_number and index of a DP Slave. Usually the slot\_number refers to a physical device in a modular slave and the index identifies an object within this module.

The confirmation of this service contains the actually written number of bytes.

### Service-Description-Block for Request:

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP
USIGN8	service	DP_WRITE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

### Data block for Request:

Data structure	T_DP_WRITE_REQ	
USIGN8	rem_add	remote address (not evaluated on MSAC_C2 connections)
USIGN8	slot_number	slot number (0..254)
USIGN8	index	index (0..254)
USIGN8	length	data length (0..240)
OCTET	data[length]	data to write

### Service-Description-Block for Confirmation:

USIGN16	comm_ref	0.. DP_MAX_CHANNELS_MSAC2-1
USIGN8	layer	DP_USR
USIGN8	service	DP_WRITE
USIGN8	primitive	CON
INT8	invoke_id	unchanged
INT16	result	POS / NEG

### Data block for Confirmation:

result = POS:

Data structure	T_DP_WRITE_CON	
USIGN16	status	(E_DP_OK)
USIGN8	rem_add	remote address (0..126)
USIGN8	slot_number	slot number (0..254)
USIGN8	index	index (0..254)
USIGN8	length	written data length

result = NEG

Data structure	T_DP_ERROR_CON	standard DP error structure (see chapter 5 about Error Structure and Error Codes)
----------------	----------------	--------------------------------------------------------------------------------------

### NOTES:

- The requested number of bytes should equal the number of bytes the slave's object will need. Otherwise, the slave might refuse to receive the data (negative confirmation with error code E\_DP\_UE).
- While the master is waiting for the response, no further requests can be issued and will be rejected with interface error E\_IF\_NO\_PARALLEL\_SERVICES, except the DP\_ABORT service.
- If the connection is closed, the service will be rejected with interface error E\_IF\_SERVICE\_NOT\_EXECUTABLE.
- The rem\_add, slot\_number and index in the confirmation are redundant informations (mirroring the request's data) and need not be evaluated. The number of bytes actually received is an important information for the user. If the length is zero, then the user should check the object's data length.

## 5 NEGATIVE CONFIRMATIONS, ERROR- AND RETURN CODES

### 5.1 NEGATIVE CONFIRMATIONS

For all negative service confirmations (error confirmations) the following standard DP error structure is used:

Data structure:	T_DP_ERROR_CON	
USIGN16	status	status
USIGN8	rem_add	remote address (0..126)
USIGN8	error_decode	error class
USIGN8	error_code_1	error code 1
USIGN8	error_code_2	error code 2

In addition to DP services, also each DP/V1 service confirmation returns an USIGN16 status value which is defined as follows:

- **USIGN8 (low byte):** Contains the error code. This byte indicates general errors like "Invalid Parameter", etc.
- **USIGN8 (high byte):** This byte is used only if an error extension is available. By means of this byte additional error or status information are reported to the user.

**Example:** STATUS = 0x2315 ⇒ E\_DP\_ILLEGAL\_INDEX and E\_DP\_IV

#### Error Codes

E_DP_OK	0x00	OK, acknowledgement positive
E_DP_UE	0x01	remote user error, see Detail Error Codes
E_DP_IV	0x15	invalid parameter in request, see Error Code
Extensions		
E_DP_SE context	0xC9 (see DP manual)	sequence error, i.e. the service is used in a wrong

#### Error Code Extension

E_DP_ILLEGAL_INDEX	0x23	illegal index in DP_READ or DP_WRITE request
E_DP_ILLEGAL_SLOT	0x24	illegal slot number in DP_READ or DP_WRITE request
E_DP_ILLEGAL_LENGTH	0x25	illegal data length in DP_READ or DP_WRITE request
E_DP_WRONG_SLAVE_ADD	0x03 (see DP manual)	invalid slave address in DP_INITIATE request
E_DP_ILLEGAL_EXTENSION	0x26	invalid network address in DP_INITIATE request



## Detail Error Codes

The parameters `error_decode`, `error_code_1` and `error_code_2` provide detailed information about the occurred error.

The parameter `error_decode` defines the meaning of the parameters `error_code_1` and `error_code_2`.

### Error\_Decode:

DP_ERROR_DECODE_DPV1	0x80	DP/V1 protocol error
DP_ERROR_DECODE_FMS	0xFE	PROFIBUS FMS protocol error
DP_ERROR_DECODE_HART	0xFF	HART protocol error

For DP/V1 the parameter `error_code_1` defines the classification of the error. The parameter `error_code_2` is slave user specific.

### Error\_Code\_1:

DP_ERROR_APP_READ	0xA0	read error
DP_ERROR_APP_WRITE	0xA1	write error
DP_ERROR_APP_MODUL_FAILURE	0xA2	module failure
	0xA3..0xA7	reserved
DP_ERROR_APP_VERSION_CONFLICT	0xA8	version conflict
DP_ERROR_APP_FEATURE	0xA9	feature not supported
	0xAA..0xAF	slave user specific
DP_ERROR_ACCESS_INVALID_INDEX	0xB0	invalid index
DP_ERROR_ACCESS_INVALID_LENGTH	0xB1	length error
DP_ERROR_ACCESS_INVALID_SLOT	0xB2	invalid slot
DP_ERROR_ACCESS_TYPE_CONFLICT	0xB3	type conflict
DP_ERROR_ACCESS_INVALID_AREA	0xB4	invalid area
DP_ERROR_ACCESS_STATE_CONFLICT	0xB5	state conflict
DP_ERROR_ACCESS_DENIED_ACCESS	0xB6	access denied
DP_ERROR_ACCESS_INVALID_RANGE	0xB7	invalid range
DP_ERROR_ACCESS_INVALID_PARAM	0xB8	invalid parameter
DP_ERROR_ACCESS_INVALID_TYPE	0xB9	invalid type
	0xBA..0xBF	slave user specific
DP_ERROR_RES_READ_CONFLICT	0xC0	read constraint conflict
DP_ERROR_RES_WRITE_CONFLICT	0xC1	write constarint conflict
DP_ERROR_RES_BUSY	0xC2	resource busy
DP_ERROR_RES_UNAVAILABLE	0xC3	resource unavailable
	0xC4..0xC7	reserved
	0xC8..0xCF	slave user specific

## 5.2 RETURN CODES

The following return codes may occur specifically for DP/V1 when a service request will be rejected. The values for these codes can be found in the User Interface manual chapter Interface Errors.

Return Code / Services	DP_INITIATE	DP_READ	DP_WRITE	DP_ABORT
E_IF_SERVICE_NOT_EXECUTABLE	x	x	x	x
E_IF_INVALID_COMM_REF	x	x	x	x
E_IF_INVALID_PRIMITIVE	x	x	x	x
E_IF_INVALID_PARAMETER	-	-	-	x
E_IF_NO_PARALLEL_SERVICES	x	x	x	-
E_IF_SERVICE_CONSTR_CONFLICT	x	-	-	-

Explanation of the return codes:

E_IF_SERVICE_NOT_EXECUTABLE	either - the master is still offline, - FMS is running parallel to DP/V1, - DP_READ or DP_WRITE services are executed on a closed connection.
E_IF_INVALID_COMM_REF	comm_ref is $\geq$ DP_MAX_CHANNELS_MSAC2
E_IF_INVALID_PRIMITIVE	something other than a request is issued to DP/V1
E_IF_INVALID_PARAMETER	an open connection is aborted with subnet out of range
E_IF_NO_PARALLEL_SERVICES	some service is already active on this comm_ref
E_IF_SERVICE_CONSTR_CONFLICT	temporarily, no connection can be opened

# **PROFIBUS Application Program Interface**

## **FDL Services**

Version 5.2  
Rev. 00

Date: 17-October-1997

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---

---

## CONTENTS

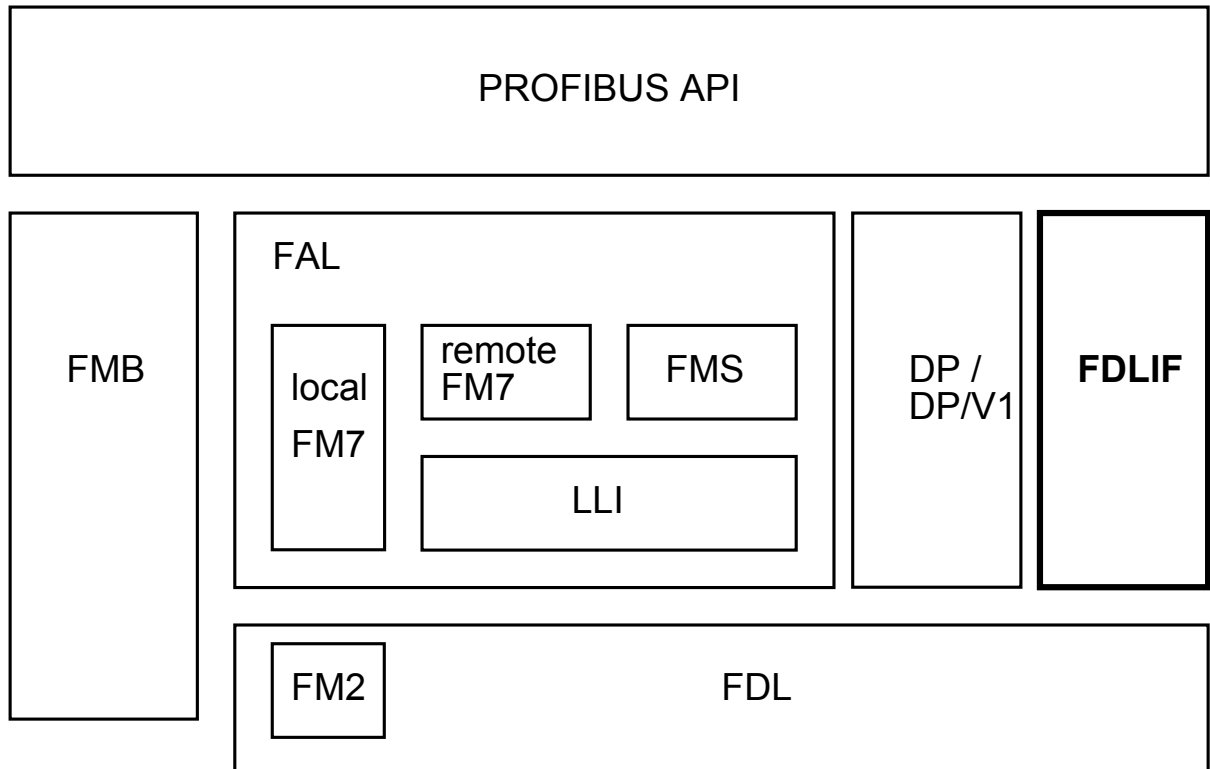
1 SCOPE .....	1
2 OVERVIEW .....	2
3 FDLIF MANAGEMENT SERVICES .....	4
3.1 FDLIF-Set-Busparameter .....	4
3.2 FDLIF-Read-Busparameter .....	6
3.3 FDLIF-Activate-SAP .....	8
3.4 FDLIF-Activate-RSAP .....	12
3.5 FDLIF-Change-SAP-Access .....	14
3.6 FDLIF-Deactivate-SAP .....	15
3.7 FDLIF-Event .....	16
3.8 FDLIF-Exit .....	17
4 FDLIF DATA TRANSFER SERVICES .....	18
4.1 FDLIF-SDA (Send Data with Acknowledge) .....	18
4.2 FDLIF-SDN (Send Data With No Acknowledge) .....	20
4.3 FDLIF-SRD (Send and Request Data with reply) .....	22
4.4 FDLIF-Reply-Update .....	25
4.5 FDLIF-Reply-Update-Multiple .....	27
APPENDIX A .....	29



## 1 SCOPE

For some applications it may be useful to have direct access to the FDL functionality. The Fieldbus Data Link Layer Interface (FDLIF) provides an interface for using FDL services excluding other protocol components. This manual describes the services that are provided by the FDLIF.

The following figure shows how the FDLIF is embedded in Softing's PROFIBUS protocol software.



Softing's PROFIBUS API provides uniform access to all service groups of the PROFIBUS protocol. The common access functions are described in the "User Interface" part of the PROFIBUS User Manual.

This document should be read in conjunction with the following parts of the PROFIBUS User Manual:

- "User Interface" (describes the uniform access functions to all PROFIBUS services)
- "FMB Services" (describes the management services which are necessary to configure the PROFIBUS protocol stack)

## 2 OVERVIEW

The FDLIF has to be activated and configured by means of the FMB-Set-Configuration service. Prior to activation and configuration no FDLIF service is usable.

FDLIF provides management services and data transfer services.

The main goal of management services is to configure FDL and to activate FDL Service Access Points (FDL SAPs). Management services cannot be executed in parallel. This means, issuing a management service request is only allowed if there is no outstanding management service confirmation. Multiple requests are rejected by FDLIF.

All data transfer services have to be processed via FDL SAPs. Before FDL SAPs may be used for data transfer, they have to be activated. Data transfer services can be executed in parallel. This means, the user is allowed to issue multiple data transfer requests without waiting for confirmations. The maximum number of outstanding confirmations is determined by "credits" that have been specified in the FMB-Set-Configuration request.

### FDLIF services in Service Group Order

#### Management Services

service group	Identifier	Code	Page
Set and read the FDL Bus Parameters	FDLIF_SET_BUSPARAMETER	5	4
	FDLIF_READ_BUSPARAMETER	6	6
Manage FDL SAP's	FDLIF_SAP_ACTIVATE	7	8
	FDLIF_RSAP_ACTIVATE	8	12
	FDLIF_SAP_CHANGE_ACCESS	9	14
	FDLIF_SAP_DEACTIVATE	10	15
FM2 Event messages	FDLIF_EVENT	19	16
Halt and restart FDLIF	FDLIF_EXIT	21	17

#### Data Transfer Services

service group	Identifier	Code	Page
Data transfer	FDLIF_SDA	0	18
	FDLIF_SDN	1	20
	FDLIF_SRD	2	22
	FDLIF_REPLY_UPDATE	3	25
	FDLIF_REPLY_UPDATE_MULTIPLE	4	27

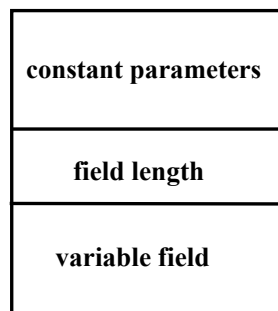


## Notes on Data Structures and Parameters

The FDLIF-specific types and constants are defined in the include file PB\_FDL.H.

All words, long words, strings, arrays and records begin on even addresses. To accomplish this, fill bytes had to be added in some places. They are always recognizable by the name *dummy*.

Data blocks do not contain pointers. If a data block contains a field or list of variable length, then the length information of all variable-length fields is stored in the constant part. The field of variable length follows on the constant part.



In the include file PB\_FDL.H variable data fields are shown between comment delimiters to show their position and structure, without forcing the programmer to use data structures of a specific length. Nevertheless, the data must be entered at exactly this spot.

The request and indication data blocks are identical.

The service description block contains a *result* parameter. If a function returns as positive (result = POS) the service-specific confirmation block will be passed. If the result is negative (result = NEG) the standard error structure T\_FDLIF\_ERROR or a service-specific data structure is passed.

The standard error structure is not noted explicitly for each service. Error structure and error codes are described in appendix A.

### 3 FDLIF MANAGEMENT SERVICES

#### 3.1 FDLIF-Set-Busparameter

This service is used to set all FDL operational parameters that are necessary to start FDL. This set of operational parameters is called FDL Bus Parameters.

The FDLIF-Set-Busparameter service has to be executed immediately after the execution of the FMB-Set-Configuration service.

Notes:

In future releases of SOFTING's PROFIBUS API, the service FDLIF-Set-Busparameter will be replaced by service FMB-Set-Busparameter. This service is provided only for compability with former releases of PROFIBUS API. Do not use this service in new applications.

This service is applicable only if FDLIF runs in stand alone mode. If a mixed operation mode has been chosen ((FDLIF and DP, DP/V1) or (FDLIF and FMS)), the FDL Bus Parameters have to be set by the FMB-Set-Busparameter service.

##### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_SET_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

##### *Data Block for Request:*

Data structure	T_FDLIF_SET_BUSPARAMETER	
USIGN8	loc_add	local station address
USIGN8	loc_segm	local segment
USIGN8	baud_rate	baud rate
USIGN8	medium_red	medium redundancy
USIGN16	tsl	slot time
USIGN16	min_tsdr	min. station delay time resp.
USIGN16	max_tsdr	max. station delay time resp.
USIGN8	tqui	quiet time
USIGN8	tset	setup time
USIGN32	ttr	target token rotation time
USIGN8	g	gap update factor
PB_BOOL	in_ring_desired	active or passive station
USIGN8	hsa	highest station address
USIGN8	max_retry_limit	max. retry limit
USIGN16	reserved	for internal use
USIGN8	ident[202]	the identification string is filled by the protocol stack

A detailed description of the FDL Bus Parameters is given in the FMB manual.

**Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured or another management service is in progress

**Service-Description-Block for Confirmation:**

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SET_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

**Data block for Confirmation:**

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

**Values of result:**

FDL_IV	invalid parameter in request
--------	------------------------------

**Values of add\_detail:**

FALSE_TS	0x0001
FALSE_BAUD_RATE	0x0002
FALSE_MEDIUM_RED	0x0004
FALSE_IDENT	0x0008
FALSE_TSL	0x0010
FALSE_MIN_TSDR	0x0020
FALSE_MAX_TSDR	0x0040
FALSE_TQUI	0x0080
FALSE_TSET	0x0100
FALSE_TTR	0x0200
FALSE_G	0x0400
FALSE_IN_RING_DESIRE	0x0800
FALSE_HSA	0x1000
FALSE_RETRY_CTR	0x2000
FALSE_STATION_TYPE	0x4000

## 3.2 FDLIF-Read-Busparameter

This service is used to read the FDL Bus Parameters.

Note:

In future releases of SOFTING's PROFIBUS API, the service FDLIF-Read-Busparameter will be replaced by service FMB-Read-Busparameter. This service is provided only for compability with former releases of PROFIBUS API. Do not use this service in new applications.

### Service-Description-Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_READ_BUSPARAMETER
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

### Return values for request:

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configurated or another management service is in progress.

### Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_READ_BUSPARAMETER
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

### Data block for Confirmation:

result = POS:

Data structure	T_FDLIF_READ_BUSPARAMETER	
USIGN8	loc_add	local station address
USIGN8	loc_segm	local segment
USIGN8	baud_rate	baud rate
USIGN8	medium_red	medium redundancy
USIGN16	tsl	slot time
USIGN16	min_tsdr	min. station delay time resp.
USIGN16	max_tsdr	max. station delay time resp.
USIGN8	tqui	quiet time
USIGN8	tset	setup time
USIGN32	ttr	target token rotation time
USIGN8	g	gap update factor
PB_BOOL	in_ring_desired	active or passive station
USIGN8	hsa	highest station address
USIGN8	max_retry_limit	max. retry limit
USIGN16	reserved	for internal use
USIGN8	ident[202]	ident

A detailed description of the FDL Bus Parameters is given in the FMB manual.

result = NEG:

Data structure

T\_FDLIF\_ERROR

error structure

## 3.3 FDLIF-Activate-SAP

All data transfer services have to be processed via FDL Service Access Points (FDL SAPs). There are two SAP types reflecting the different characteristics of Master Stations and Slave Stations. Usually, "regular" SAPs are activated on Master Stations whereas Responder SAPs (RSAPs) are activated on Slave Stations. However, it is allowed to use RSAPs on Master Stations and "regular" SAPs on Slave Stations.

The service FDLIF-Activate-SAP is used to activate and configure "regular" SAPs.

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_SAP_ACTIVATE
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

### *Data Block for Request:*

Data structure	T_FDLIF_SAP_ACTIVATE_REQ	
USIGN8	sap_nr	SAP to be activated (0..63,FDL_DEFAULT_SAP)
USIGN8	max_l_sdu_length_req	maximum length of request telegram
USIGN8	max_l_sdu_length_con_ind	maximum length of con/ind telegram
USIGN8	access_sap	permitted request SAPs
USIGN8	access_station	permitted requestor
USIGN8	sda	role in SDA service
USIGN8	sdn	role in SDN service
USIGN8	srđ	role in SRD service
USIGN8	csrd	not supported, SERVICE_NOT_ACTIVATED
USIGN8	data_mode	not used
USIGN8	credits	number of indication resources
USIGN8	dummy	alignment byte

### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured or another management service is in progress.
E_IF_RESOURCE_UNAVAILABLE	21	no resource available for service processing, i.e all receive_credits are used

### Notes on data structure T\_FDLIF\_SAP\_ACTIVATE\_REQ

The component *sap\_nr* specifies the number of the SAP that should be activated. Valid values are 0..63 and FDL\_DEFAULT\_SAP.

The component *max\_l\_sdu\_length\_req* specifies the maximum length of user data that can be sent by this SAP. The upper bound of *max\_l\_sdu\_length\_req* depends on the local SAP number and the remote SAP number in the data transfer request.

local SAP	remote SAP	range of <i>max_l_sdu_length_req</i>
not FDL_DEFAULT_SAP	not FDL_DEFAULT_SAP	0..244
not FDL_DEFAULT_SAP	FDL_DEFAULT_SAP	0..245
FDL_DEFAULT_SAP	not FDL_DEFAULT_SAP	0..245
FDL_DEFAULT_SAP	FDL_DEFAULT_SAP	0..246

The component *max\_l\_sdu\_length\_con\_ind* specifies the maximum length of user data that can be received by this SAP. The upper bound of *max\_l\_sdu\_length\_con\_ind* depends on the local SAP number and the remote SAP number in data confirmations and indications.

local SAP	remote SAP	range of <i>max_l_sdu_length_con_ind</i>
not FDL_DEFAULT_SAP	not FDL_DEFAULT_SAP	0..244
not FDL_DEFAULT_SAP	FDL_DEFAULT_SAP	0..245
FDL_DEFAULT_SAP	not FDL_DEFAULT_SAP	0..245
FDL_DEFAULT_SAP	FDL_DEFAULT_SAP	0..246

The component *access\_sap* with the values 0..62, FDL\_DEFAULT\_SAP and ALL is used for access protection. It specifies whether request PDUs are accepted only from a single remote SAP (0..62, FDL\_DEFAULT\_SAP) or from all remote SAPs (ALL).

Just as *access\_sap* the component *access\_station* with the values 0..126 and ALL is used for access protection. It specifies whether request PDUs are accepted only from a single remote station (0..126) or from all remote stations (ALL).

The components *sda*, *sdn*, *srd* and *csrd* specifies the services that are activated for this SAP. For each service there are different roles. The following roles are possible:

- INITIATOR: The station initiates the service exclusively
- RESPONDER: The station responds to the service exclusively
- BOTH\_ROLES: The station initiates and responds to the service.
- SERVICE\_NOT\_ACTIVATED: The service is not active for this SAP.

If the SAP is activated on a Master Station the following combinations of service and role are possible:

	INITIATOR	RESPONDER	BOTH_ROLES	SERVICE_NOT_ACTIVATED
SDA	x	x <sup>(1)</sup>	x <sup>(1)</sup>	x
SDN	x	x <sup>(1)</sup>	x <sup>(1)</sup>	x
SRD	x	x <sup>(1)</sup>	x <sup>(1)</sup>	x
CSRD				x

- (1) The ASPC2 does not accept to activate a SAP as RESPONDER for two of the services SDA, SDN or SRD. So either all three services are activated as RESPONDER or a single service is activated as RESPONDER.

If the SAP is activated on a Slave Station only one combination of service and role is possible:

	INITIATOR	RESPONDER	BOTH_ROLES	SERVICE_NOT_ACTIVATED
SDA				x
SDN		x		
SRD				x
CSRD				x

The component *data\_mode* is used for RSAPs only.

A SAP needs data buffers (receive resources) to receive PDUs from remote stations. The component *credits* specifies the number of receive resources that are assigned to the SAP.

There is a time gap between the receipt of a PDU in the FDL and the indication on the user interface. Thus, indications are stored in the protocol stack temporarily. The component *credits* specifies the number of indications that may be stored in the protocol stack, i.e. *credits* specifies how many indications may be processed by the protocol stack in parallel. Note that the sum of credits for all SAPs must not exceed the number of receive credits that have been specified in the FMB-Set-Configuration request.



### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SAP_ACTIVATE
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

### **Values of result:**

FDL_IV	invalid parameter in request
FDL_NO	SAP already active

### 3.4 FDLIF-Activate-RSAP

This service is used to activate and configure a FDL Responder SAP (FDL RSAP). (See also service FDLIF-Activate-SAP.)

Note that the number of active RSAPs must not exceed the maximum number of Responder SAPs that have been specified in the FMB-Set-Configuration request.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_RSAP_ACTIVATE
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

#### *Data Block for Request:*

Data structure	T_FDLIF_RSAP_ACTIVATE_REQ	
USIGN8	sap_nr	SAP to be activated (0..63,FDL_DEFAULT_SAP)
USIGN8	max_l_sdu_length_req	maximum length of request telegram
USIGN8	max_l_sdu_length_ind	maximum length of ind telegram
USIGN8	access_sap	permitted request SAPs
USIGN8	access_station	permitted requestor
USIGN8	data_mode	NORMAL_MODE, DATA_MODE
USIGN8	credits	number of indication resources
USIGN8	dummy	alignment byte

#### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configurated or another management service is in progress.
E_IF_RESOURCE_UNAVAILABLE	21	no resource available for service processing, i.e either all receive credits are used or all Responder SAPs are activated.

### Notes on data structure T\_FDLIF\_RSAP\_ACTIVATE\_REQ

Most of the components are analogous to the data structure T\_FDLIF\_SAP\_ACTIVATE\_REQ. Differences and add-ons are as follows:

The roles for the services SDA, SDN, SRD and CSRD are implicitly set. The roles are set as follows:

	RESPONDER	SERVICE_NOT_ACTIVATED
SDA		x
SDN		x
SRD	x	
CSRD		x

The component *data\_mode* specifies whether a SRD indication shall be generated for all received SRD request PDUs (NORMAL\_MODE), or the SRD indication shall be omitted if both the received request PDU and the sent response PDU contain no user data (DATA\_MODE).

#### Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_RSAP_ACTIVATE
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

#### Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

#### Values of result:

FDL_IV	invalid parameter in request
FDL_NO	SAP already activated

## 3.5 FDLIF-Change-SAP-Access

When a SAP is activated the components *access\_sap* and *access\_station* of T\_FDLIF\_SAP\_ACTIVATE\_REQ and T\_FDLIF\_RSAP\_ACTIVATE\_REQ specifies the access rights of remote stations. The service may be used to change the access rights of an activated (R)SAP.

### Service-Description-Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_SAP_CHANGE_ACCESS
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

### Data Block for Request:

Data structure	T_FDLIF_SAP_CHANGE_REQ	
USIGN8	sap_nr	SAP to be activated (0..63, FDL_DEFAULT_SAP)
USIGN8	access_sap	0..62, FDL_DEFAULT_SAP or ALL
USIGN8	access_station	0..126 or ALL
USIGN8	dummy	alignment byte

### Return values for request:

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configurated or another management service is in progress.

### Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SAP_CHANGE_ACCESS
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

### Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

### Values of result:

FDL_IV	invalid parameter in request
FDL_NO	SAP is not active

### 3.6 FDLIF-Deactivate-SAP

This service is used to deactivate an FDL (R)SAP.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_SAP_DEACTIVATE
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

#### *Data Block for Request:*

Data structure	T_FDLIF_SAP_DEACTIVATE_REQ	
USIGN8	sap_nr	SAP to be deactivated (0..63,FDL_DEFAULT_SAP)
USIGN8	dummy	alignment byte

#### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured or another management service is in progress.

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SAP_DEACTIVATE
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

#### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

#### **Values of result:**

FDL_IV	invalid parameter in request
FDL_NO	SAP is not active

## 3.7 FDLIF-Event

By means of the FDLIF-Event service, the FDLIF indicates FM2 events to the FDLIF user.

### Notes:

In the FMB-Set-Configuration service the user selects the instance that shall receive the FM2 events. If the FDLIF user is selected as event receiver, the FM2 events are indicated as FDLIF-Events.

In future releases of SOFTING's PROFIBUS API, the service FDLIF-Event will be replaced by service FMB-Event. This service is provided only for compatibility with former releases of PROFIBUS API. Do not use this service in new applications.

### Service-Description-Block for die Indication:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_EVENT
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	POS

### Data Block:

Data structure	T_FDLIF_EVENT_IND	
USIGN8	event	event code

### FM2 event codes:

FM2_FAULT_ADDRESS	1	duplicate address recognized	
FM2_FAULT_PHY	2	physical layer is malfunctioning	(1)
FM2_FAULT_TTO	3	timeout on bus detected	
FM2_FAULT_SYN	4	no receiver synchronization	
FM2_FAULT_OUT_OF_RING	5	local station out of ring	
FM2_GAP_EVENT	6	GAP area has changed	(1)

(1) Not supported by ASPC2

### Additional FM2 event codes (Error messages from ASPC2)

FM2_MAC_ERROR	19	fatal MAC error
FM2_HW_ERROR	20	fatal HW error

### 3.8 FDLIF-Exit

The FDLIF-Exit service may be used to reset the FDLIF component. Reset of FDLIF component means loss of all outstanding confirmations, deactivation of all SAPs activated via FDLIF, and - if FDLIF runs in stand alone mode - reset of FDL which includes reset of ASPC2.

*Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_EXIT
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	POS

*Data block for Request:*

n/a

*Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_EXIT
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS

*Data block for Confirmation:*

n/a

## 4 FDLIF DATA TRANSFER SERVICES

### 4.1 FDLIF-SDA (Send Data with Acknowledge)

This service is used to send data to a single remote station. At the remote station the data - if received error-free - are passed to the remote FDL user. The local FDLIF user gets a confirmation concerning the receipt or non-receipt of the data.

#### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF, FDLIF_USR
USIGN8	service	FDLIF_SDA
USIGN8	primitive	REQ, IND
INT8	invoke_id	reserved for user
INT16	result	POS

#### *Data Block for Request and Indication:*

Data structure	T_FDLIF_SDN_SDA_SRD_REQ	
USIGN8	ssap	source SAP (0..62)
USIGN8	dsap	destination SAP at remote station (0..62)
USIGN8	rem_add	address of remote station (0..126)
USIGN8	priority	priority (LOW, HIGH)
USIGN8	status	only used for SRD indications
USIGN8	length	length of request/indication data
USIGN8	req_data[length]	request/indication data

#### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured
E_IF_RESOURCE_UNAVAILABLE	21	all send credits are in use

#### **Notes on T\_FDLIF\_SDN\_SDA\_SRD\_REQ**

The maximum length of data that may be sent or received has been specified by means of the FDLIF-Activate-SAP service.



## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SDA
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

## Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

## Values of result:

FDL_UE	user error at remote station
FDL_RR	resource at remote SAP not available or not sufficient
FDL_RS	service at remote SAP not activated or remote SAP not activated
FDL_NA	no reaction from remote station
FDL_DS	local FDL is not in logical token ring
FDL_LS	service at local SAP or local SAP not activated
FDL_IV	invalid parameter in request

## 4.2 FDLIF-SDN (Send Data With No Acknowledge)

This service is used to send data to a single remote station, to a group of remote stations (Multicast) or to all remote stations (Broadcast). At the remote station(s) the data - if received error free - are passed to the remote FDL user(s). The local FDLIF user gets a confirmation acknowledging the end of data transfer. However, there is no confirmation that a successful receipt of data has taken place.

### *Service-Description-Block for Request and Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF, FDLIF_USR
USIGN8	service	FDLIF_SDN
USIGN8	primitive	REQ, IND
INT8	invoke_id	reserved for user
INT16	result	POS

### *Data Block for Request and Indication:*

Data structure	T_FDLIF_SDN_SDA_SRD_REQ	
USIGN8	ssap	source SAP (0..62)
USIGN8	dsap	destination SAP at remote station (0..62,63)
USIGN8	rem_add	address of remote station (0..126,127)
USIGN8	priority	priority (LOW, HIGH)
USIGN8	status	only used for SRD indications
USIGN8	length	length of request/indication data
USIGN8	req_data[length]	request/indication data

### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured
E_IF_RESOURCE_UNAVAILABLE	21	all send credits are in use

### **Notes on T\_FDLIF\_SDN\_SDA\_SRD\_REQ**

The type of a SDN request depends on remote address and remote SAP:

message type	remote address	remote SAP (dsap)
SDN to a single remote station	0..126	0..62
SDN Broadcast	127	63
SDN Multicast	127	0..62

The maximum length of data that may be sent or received has been specified by means of the FDLIF-ACTIVATE-SAP service.

### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SDN
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

### *Data block for Confirmation:*

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

### **Values of result:**

FDL_DS	local FDL is not in logical token ring
FDL_LS	service at local SAP or local SAP not activated
FDL_IV	invalid parameter in request

### 4.3 FDLIF-SRD (Send and Request Data with reply)

This service is used to transfer data to a single remote station and at the same time to request data that was made available by the remote user at an earlier time. At the remote station the received data - if received error-free - is passed to the remote user. The local user gets either the requested data or a confirmation that remote data were not available or a confirmation of the non-receipt of the transmitted data.

The receiver SAP at the remote station has to be a FDL Responder SAP (RSAP) and the remote user has to load his data to the RSAP using the services FDLIF-REPLY-UPDATE or FDLIF-REPLY-UPDATE-MULTIPLE.

The FDLIF-SRD service can be used to request data from the remote user without sending own data. For this purpose the data length in the SRD request has to be set to zero.

#### *Service-Description-Block for Request and Indication*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF, FDLIF_USR
USIGN8	service	FDLIF_SRD
USIGN8	primitive	REQ, IND
INT8	invoke_id	reserved for user
INT16	result	POS

#### *Data Block for Request and Indication:*

Data structure	T_FDLIF_SDN_SDA_SRD_REQ	
USIGN8	ssap	source SAP
USIGN8	dsap	destination SAP of remote station
USIGN8	rem_add	address of remote station
USIGN8	priority	priority (LOW, HIGH)
USIGN8	status	SRD indication: FDL_NO_DATA, FDL_LOW_DATA, FDL_HIGH_DATA
USIGN8	length	length of request/indication data
USIGN8	req_data[length]	request/indication data

#### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured
E_IF_RESOURCE_UNAVAILABLE	21	all send credits are in use

### Notes on T\_FDLIF\_SDN\_SDA\_SRD\_REQ

Only RSAPs may be used as SRD responder.

The maximum length of data that may be sent or received has been specified by means of the FDLIF-Activate-(R)SAP service.

In a SRD indication the *status* byte indicates whether or not response data were sent to the SRD requester.

Possible values:

- FDL\_NO\_DATA: No response data were transmitted
- FDL\_LOW\_DATA: Low prior response data were transmitted
- FDL\_HIGH\_DATA: High prior response data were transmitted

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_SRD
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

## Data block for Confirmation:

result = POS:

Data structure	T_FDLIF_SRD_CNF	
USIGN8	status	status
USIGN8	length	length of confirmation data
USIGN8	cnf_data[length]	confirmation data

### Values of status:

FDL_DL	pos. ack. for sent data, reply data with low priority available
FDL_DH	pos. ack. for sent data, reply data with high priority available
FDL_NR	pos. ack. for sent data, reply data are not available from remote FDL

result = NEG:

Data structure	T_FDLIF_SRD_CNF	
USIGN8	status	status
USIGN8	length	length of confirmation data
USIGN8	cnf_data[length]	confirmation data

### Values of status:

FDL_UE	user error at remote station
FDL_RR	resource at remote SAP not available or not sufficient
FDL_RS	service at remote SAP not activated or remote SAP not activated
FDL_NA	no reaction from remote station
FDL_DS	local FDL is not in logical token ring
FDL_LS	service at local SAP or local SAP not activated
FDL_IV	invalid parameter in request
FDL_RDL	neg. ack. for sent data as resources at remote station not available or not sufficient, reply data with low priority available
FDL_RDH	neg. ack. for sent data as resources at remote station not available or not sufficient, reply data with high priority available

## 4.4 FDLIF-Reply-Update

This service allows the user to load data in the update buffer of a FDL RSAP. The user gets a REPLY-UPDATE confirmation acknowledging that the data have been stored in the buffer. The data remain in the buffer until they are transmitted as response of a SRD request PDU. The buffer contents are transmitted only one time: if the RSAP receives another SRD request PDU, it acknowledges with no data.

In the SRD indication a status byte informs the user whether or not response data were sent to the SRD requester.

If the user loads new data before the old data are transmitted the old buffer contents are overwritten!

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_REPLY_UPDATE
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

### *Data Block for Request:*

Data structure	T_FDLIF_RUP_REQ	
USIGN8	sap_nr	local RSAP
USIGN8	priority	priority (LOW, HIGH)
USIGN8	dummy	alignment byte
USIGN8	length	length of request data
USIGN8	req_data[length]	request data

### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configurated
E_IF_RESOURCE_UNAVAILABLE	21	all send credits are in use

### **Notes on T\_FDLIF\_RUP\_REQ**

The maximum length of data that may be sent has been specified by means of the FDLIF-Activate-RSAP service.

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_REPLY_UPDATE
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

## Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

## Values of result:

FDL_LS	service at local SAP or local SAP not activated
FDL_LR	resource at local SAP not available or not sufficient
FDL_IV	invalid parameter in request



## 4.5 FDLIF-Reply-Update-Multiple

This service allows the user to load data to the update buffer of a FDL RSAP. The user gets a FDLIF-REPLY-UPDATE-MULTIPLE confirmation acknowledging that the data have been stored in the buffer. The data are transmitted as response to SRD request PDU. In contrast to the FDLIF-REPLY-UPDATE service, the data are transmitted multiple times. Every receipt of a SRD request telegram is acknowledged by sending the buffer contents.

In the SRD indication a status byte informs the user whether or not response data were sent to the SRD requester.

The data are sent until the user loads new data to the update buffer. The buffer contents may be deleted by loading an empty data buffer. For this purpose the data length in the FDLIF-REPLY-UPDATE-MULTIPLE request shall be set to zero.

### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF
USIGN8	service	FDLIF_REPLY_UPDATE
USIGN8	primitive	REQ
INT8	invoke_id	reserved for user
INT16	result	POS

### *Data Block for Request:*

Data structure	T_FDLIF_RUP_REQ	
USIGN8	sap_nr	local sap SAP
USIGN8	priority	priority (LOW, HIGH)
USIGN8	dummy	alignment byte
USIGN8	length	length of request data
USIGN8	req_data[length]	request data

### **Return values for request:**

E_OK	0	request was accepted
E_IF_SERVICE_CONSTR_CONFLICT	23	system memory is not configured or an-management service is in progress.
E_IF_RESOURCE_UNAVAILABLE	21	all send credits are in use

### **Notes on T\_FDLIF\_RUP\_REQ**

The maximum length of data that may be sent has been specified by means of the FDLIF-ACTIVATE-RSAP service.

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	FDLIF_USR
USIGN8	service	FDLIF_REPLY_UPDATE
USIGN8	primitive	CON
INT8	invoke_id	same value as set with the request
INT16	result	POS or NEG

## Data block for Confirmation:

result = POS:

n/a

result = NEG:

Data structure	T_FDLIF_ERROR	error structure
----------------	---------------	-----------------

### Values of result:

FDL_LS	service at local SAP or local SAP not activated
FDL_LR	resource at local SAP not available or not sufficient
FDL_IV	invalid parameter in request

## APPENDIX A

For negative confirmations the following standard error structure is used:

Data structure:	T_FDLIF_ERROR	
USIGN8	result	error result
USIGN8	dummy	alignment byte
INT16	add_detail	additional detail

The possible FDL-Interface service error codes are follows.

### FDL-Interface Services Error Codes

Constant	Value	Description
FDL_UE	0x01	user error at remote station
FDL_RR	0x02	resource at remote SAP not available or not sufficient
FDL_RS	0x03	service at remote SAP not activated or remote SAP not activated
FDL_RDL	0x0C	neg. ack. for sent data as resources at remote station not available or not sufficient, reply data with low priority available
FDL_RDH	0x0D	neg. ack. for sent data as resources at remote station not available or not sufficient, reply data with high priority available
FDL_LS	0x10	service at local SAP or local SAP not activated
FDL_NA	0x11	no reaction from remote station
FDL_DS	0x12	local FDL is not in logical token ring
FDL_NO	0x13	configuration error at local SAP
FDL_LR	0x14	resource at local SAP not available or not sufficient
FDL_IV	0x15	invalid parameter in request



# **PROFIBUS Application Program Interface**

## **DP Slave Services**

Version 5.2  
Rev. 00

Date: 08-April-1999

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 - 45 65 6 - 0  
Fax (++49) 89 - 45 65 6 - 399

© Copyright by Softing AG, 1998-2003  
All rights reserved.

## Copyright Notice

All rights reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice. A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors, please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product, if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1998-2003 by Softing AG, Haar

CONTENTS

1 SCOPE ..... 1

2 ABOUT DPS ..... 1

3 DPS SERVICES ..... 4

    3.1 Dps\_Init\_Slave ..... 4

    3.2 Dps\_Exit\_Slave ..... 12

    3.3 Dps\_Get\_Status ..... 13

    3.4 Dps\_Slave\_Diag ..... 17

    3.5 Dps\_Chk\_Cfg ..... 22

    3.6 Dps\_Set\_Prm ..... 24

    3.7 Dps\_Set\_Slave\_Add ..... 28





## 1 SCOPE

This documentation describes the handling of the PROFIBUS DP slave (DPS). It starts with a short introduction to the transfer protocol for PROFIBUS DP.

At the beginning we briefly explain the most important terms and functions of PROFIBUS DPS. If you have already worked in this area, you can skip this chapter.

## 2 ABOUT DPS

DPS is a freely programmable DP slave that leaves any possible freedom to the applications programmer while taking many tasks off the user's hands.

Before commissioning a DP system all stations must be assigned unique addresses. A DP slave that has not yet been assigned its own address receives the default address 126. The default address is stored in the nonvolatile RAM (NVRAM) of the slave before commissioning. If you add the slave to the bus with this address, the slave needs to be assigned a unique address by the master via 'set\_slave\_add' first. The now valid slave address is taken over into the slave's nonvolatile memory. The same applies to any subsequent address assignment over the bus. The user is informed by an indication, provided the slave has received a valid new address.

Every DP slave is assigned to one DP master. The assignment is established when parameterizing the slave. After a valid configuration, only the master that has parameterized and configured the slave can use the slave for influencing the process.

Before you can address a slave over the bus you need to initialize the slave first. In the initialization process you define whether the slave should work in a specific mode (Sync, Freeze, ...). In addition, you specify the minimum reaction time, the ident number of the device, and the maximum length parameters (resource definition) the slave should support.

The slave also features the so-called "auto-parameters": 'auto\_cfg\_response', 'auto\_prm\_response' and 'auto\_startup\_inputs' that help you initialize the slave. The functionality of these parameters relieves you of having to check the individual parameterization and configuration data. When the parameters are set to PB\_TRUE the slave tests the parameterization and configuration data automatically.

When you declare special limits or controller parameters, for example, you use the so-called vendor-specific data (user\_prm\_data) in the parameterization data. If the parameterization data of the master contain vendor-specific data, these data may need to be checked. For this purpose, set the parameter 'auto\_prm\_response' to PB\_FALSE. The parameterization data are transferred with an indication. When you have checked the data you have to either approve or reject them.

You may in some cases also need to check the configuration data. Let's assume, for example, that you would like the slave to support different configurations. You initialize the slave with a possible configuration. The master, however, transmits a different configuration. If you then check the configuration data and determine that this configuration is permitted as well, the slave takes over the new configuration that was transmitted by the master. To be able to check the configuration data transmitted by the master, you set the parameter 'auto\_cfg\_response' to PB\_FALSE.

The following section provides a short description of the slave's operating states to help you understand error messages that may occur and to become familiar with specific sequences of operation. The individual states of the slave and the associated status transitions are illustrated in the figure below.

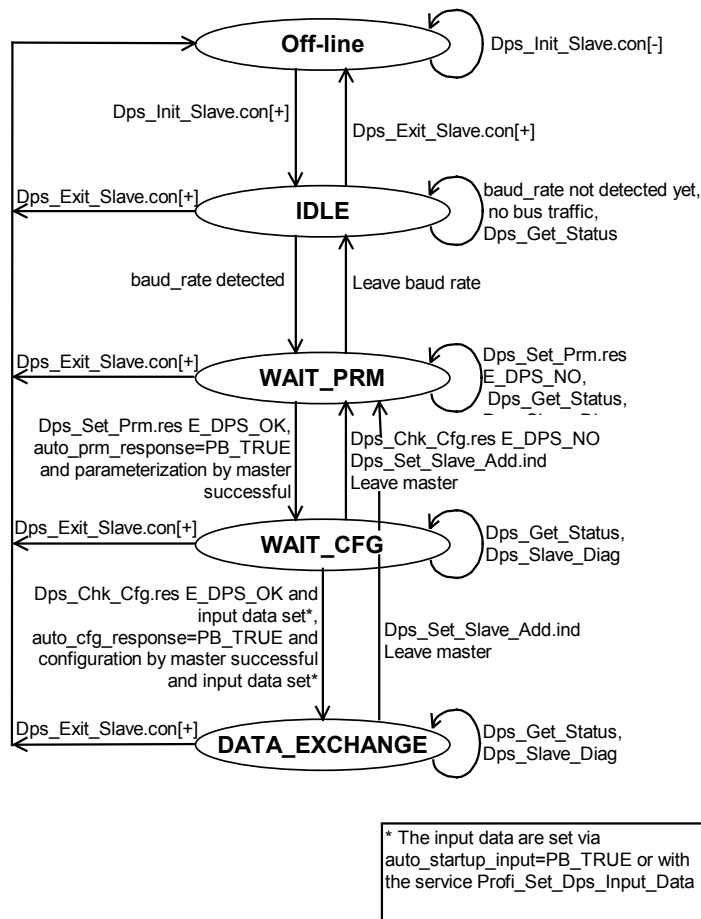


Fig-2-1: Status diagram of the DP slave

If no primitive is specified in the services shown in the figure, all service primitives are possible. In addition, the data interface functions:

- profi\_set\_dps\_input\_data
- profi\_get\_dps\_output\_data
- profi\_get\_dps\_input\_data

have not been included in the figure since they can be used in all operating states without influencing the states.

After initialization, the slave changes into the operating state IDLE. It remains in this state until the ASIC detects a valid baud rate. The slave then changes into WAIT\_PRM state and waits for parameterization data from the master. You will only be informed with a 'Dps\_Set\_Prm indication' that parameterization data were received if user\_prm\_data exist or in the case of auto\_prm\_response=PB\_FALSE. The status transition to WAIT\_CFG is independent of your response to the configuration data. Information on whether this response is still missing is stored in the parameter 'prm\_wait\_response' of the service 'Dps\_Get\_Status' and in the indication 'Dps\_Set\_Prm'.

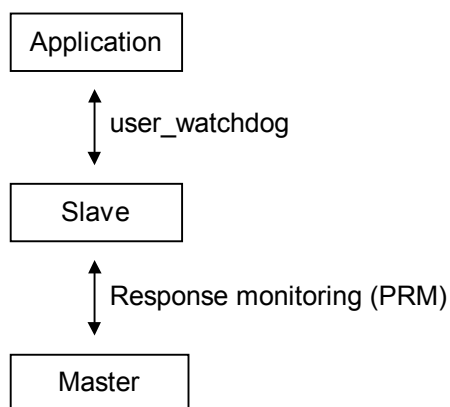
The slave changes from the operating state `WAIT_CFG` to `DATA_EXCHANGE` when both the parameterization and configuration data have been accepted by the slave or user and the first input data of the slave have been set. The operating state is changed automatically when the parameters 'auto\_cfg\_response' and 'auto\_startup\_inputs' are simultaneously set to `PB_TRUE`. If a response to the configuration is still to come from the user, the corresponding parameter 'cfg\_wait\_response' is set to `PB_TRUE`. If there are still input data missing, however, 'startup\_input\_wait' is set to `PB_TRUE`. You set the input data with the function 'profi\_set\_dps\_input\_data' of the data interface.

As soon as the slave goes through a relevant change of state, the firmware automatically generates an indication of the service 'Dps\_Get\_Status'. Changes of state are considered relevant when they have an influence on the transfer of data, i.e. a transition from or into the operating state `DATA_EXCHANGE`, Global\_Control commands or a transition of the master to `CLEAR` or `OPERATE`. Finding the baud rate, etc. is not indicated automatically. You can, of course, interrogate the slave's state yourself at any time.

Every service confirmation includes a parameter 'USIGN16 status'. This parameter contains the status which shows whether the service was executed correctly. If the confirmation has a positive result, this status is `E_DPS_OK`. If negative, it describes the reason why the service could not be executed. In addition, the high byte of the status contains an extended error code which provides additional information on the error that occurred. If, for example, the low byte shows `E_DPS_IV`, the service request contains an invalid parameter. `E_DPS_WRONG_SLAVE_ADDRESS` in the high byte may indicate that an invalid address was used.

You are not notified whether a control command (Sync or Freeze) was received. You are informed, however, that the status Sync/Freeze-Enabled/-Disabled has changed. Information on the status of the inputs and/or outputs can be interrogated through the data interface functions. The parameter 'state' is returned and describes the status of the inputs/outputs. The commands Sync/Freeze have already been handled by the ASIC and are referenced to the time the data buffer is exchanged in the ASIC. The status information is then only for your information.

The firmware offers two monitoring mechanisms. The first mechanism is used for monitoring the application (user\_watchdog\_timeout) and the second for monitoring the master (response monitoring). The response monitoring of the slave assures that, in the case of a failure of the master, the outputs change into failsafe state when the response monitoring time has run out. Application monitoring detects whether the application accesses the data interface during the monitoring time.



**Fig.2-2:** Monitoring mechanisms

### 3 DPS SERVICES

#### 3.1 Dps\_Init\_Slave

When initializing the DP slave you assign it basic functionalities (such as Sync, Set\_Slave\_Add) as well as time parameters.

The buffer of the ASIC used is limited to approx. 1.4 KB. For this reason, the memory needs to be allocated during initialization according to the maximum lengths required for input and output, configuration, parameterization, diagnostics and slave address change.

The data block of the service confirmation provides you with the value of the memory capacity that has remained or is missing after initialization.

	Buffer for inputs			Buffer for outputs		Buffer for diagnostic data	Buffer for CFG data	Buffer for PRM data	Buffer for data for set_ slave_ add	Auxil- iary buffer
1	2	3	1	2	3	1	2	1	1	1

*Fig.3.1-1: Allocation of the ASIC buffer*

The firmware provides a monitoring mechanism for the application; this mechanism is enabled with the parameter 'user\_watchdog\_timeout'. If the application does not access the data interface within the specified monitoring time, a timeout occurs. This has the following consequences:

- The slave leaves data exchange and the bits DPS\_DIAG\_BIT\_STAT\_DIAG and DPS\_DIAG\_BIT\_EXT\_DIAG are set automatically. The slave remains in the operating state DATA\_EXCHANGE, however, it no longer exchanges any data but only diagnostic telegrams.
- When you interrogate the slave's state with the service 'Dps\_Get\_Status', the state state = E\_DPS\_TO is returned. In addition, a 'Dps\_Get\_Status.ind' is generated.

The watchdog is retriggered each time the data interface is accessed. If a timeout occurred, you can reset it by setting a diagnostic (see service 'Dps\_Slave\_Diag').

The service 'Dps\_Init\_Slave' must be called before all other DPS services. If this service was not called, all requests are rejected with the error message E\_IF\_SERVICE\_NOT\_EXECUTABLE.

## Service-Description-Block for Request:

USIGN16	comm_ref	not used
USIGN8	layer	DPS
USIGN8	service	DPS_INIT_SLAVE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

## Data Block for Request:

Data structure	T_DPS_INIT_SLAVE_REQ	
USIGN8	slave_add	Station address of the slave
USIGN8	min_tsd	Minimum reaction time [T <sub>bit</sub> ]
PB_BOOL	auto_cfg_response	Checking the configuration data
PB_BOOL	auto_prm_response	Checking the parameterization data
PB_BOOL	auto_startup_inputs	Readying the first input data
PB_BOOL	sync_mode_supported	Freezing the output states
PB_BOOL	freeze_mode_supported	Freezing the inputs
PB_BOOL	set_slave_add_supported	Changing the slave address over the bus
USIGN8	max_input_data_len	Maximum length of input data [byte] (0..DP_MAX_INPUT_DATA_LEN)
USIGN8	max_output_data_len	Maximum length of output data [byte] (0..DP_MAX_OUTPUT_DATA_LEN)
USIGN8	max_cfg_data_len	Maximum length of configuration data [byte] (1..DP_MAX_CFG_DATA_LEN)
USIGN8	max_usr_prm_data_len	Maximum length of user-defined parameterization data [byte] (0..DP_MAX_USER_PRM_DATA_LEN)
USIGN8	max_ext_diag_data_len	Maximum length of user-defined diagnostic data [byte] (0..DP_MAX_EXT_DIAG_DATA_LEN)
USIGN8	max_address_data_len	Maximum length of address data [byte]
USIGN16	ident_number	Ident number
USIGN16	user_watchdog_timeout	Application monitoring time for data access [ms]
USIGN8	reserved[4]	Reserved for enhancements. Must be set to 0
USIGN8	cfg_data_len	Length of the configuration data [byte] (1..DP_MAX_CFG_DATA_LEN)
USIGN8	enhanced_init_data_len	Reserved for enhancements. Must be set to 0
USIGN8	cfg_data[DP_MAX_CFG_DATA_LEN]	Initial configuration data
USIGN8	enhanced_init_data[DPS_MIN_SERVICE_IF_LEN-DP_MAX_CFG_DATA_LEN-24]	Reserved for enhancements

## slave\_add

0..125	Slave address
DPS_DEFAULT_SLAVE_ADD (126)	The slave goes to the bus with the default address and does not exchange any data.
DPS_NON_VOLATILE_SLAVE_ADD	The slave is assigned the address that is stored in nonvolatile memory. The default address stored is 126. Otherwise, any address that the master assigns the slave with the service 'set_slave_add' is taken over into nonvolatile memory.

## NOTE:

The slave can only be commissioned with the default address **DPS\_DEFAULT\_SLAVE\_ADD** if the parameter 'set\_slave\_add\_supported' is set to **PB\_TRUE** and memory was configured for set\_slave\_add.

## min\_tsdr

The minimum reaction time is the minimum length of time the slave needs to wait before it is permitted to return its reply telegrams to the DP master.

0.. 10	Represents no change
11..255	Minimum reaction time

## auto\_cfg\_response

PB_FALSE	Configuration data transmitted by the master are passed on to the user. The user must check the configuration data and approve or reject the configuration with the service 'Dps_Chk_Cfg.res'. The parameter 'cfg_wait_response' of the service 'Dps_Get_Status' or the 'Dps_Chk_Cfg.ind' indicate whether a response to the data is still to come.
PB_TRUE	The slave's actual configuration is compared with the nominal configuration transmitted by the master. The check determines whether the information on format and length, the number of inputs and outputs and the consistency agree. The actual configuration is located in the parameter 'cfg_data'.

## auto\_prm\_response

PB_FALSE	The vendor-specific parameterization data are checked by the user and either approved or rejected. Checking the standard parameterization data is performed by the slave. The parameter 'prm_wait_response' of the service 'Dps_Get_Status' or the 'Dps_Set_Prm.ind' indicate whether a response to the data is still to come.
PB_TRUE	When the parameterization data were accepted by the slave, it automatically enters into user data exchange as soon as there are valid input data. If vendor specific parameterization data exist, they are transferred to the user without requiring a response.

## auto\_startup\_inputs

After configuration and parameterization, the slave only changes into DATA\_EXCHANGE state if the inputs have been set for the first time.

PB_TRUE	The first input data are automatically set to the value 0.
PB_FALSE	Before the slave can enter into data exchange, the user needs to write the input data for the first time. The parameter 'startup_input_wait' of the service 'Dps_Get_Status' indicates whether input data need to be transferred. The input data are set with the function 'profi_set_dps_input_data'.

## sync\_mode\_supported

PB_TRUE	The function for freezing the output data is supported by the slave.
PB_FALSE	The slave does not support the function for freezing the outputs.

### freeze\_mode\_supported

PB_TRUE	The slave supports the function for freezing the inputs.
PB_FALSE	The function for freezing the inputs is not supported by the slave.

### set\_slave\_add\_supported

PB_TRUE slave	The station address of the slave can be changed over the bus with 'set_slave_add'. The new address is stored to the slave's nonvolatile memory.
PB_FALSE	A change of the station address by the master is not supported by the slave.

### max\_address\_data\_len

If the slave supports a change of address over the bus, the maximum length of the address data must be set to at least DPS\_MIN\_SSA\_DATA\_LEN.

0	Changing the slave address over the bus is not supported.
---	-----------------------------------------------------------

DPS\_MIN\_SSA\_DATA\_LEN ..  
DP\_MAX\_TELEGRAM\_LEN

### NOTE:

The ASIC buffer is too small to initialize a slave that contains 244 bytes of input and 244 bytes of output data simultaneously. The maximum possible values are 216 bytes of input and 216 bytes of output data simultaneously. If max\_cfg\_data\_len = 8, max\_usr\_prm\_data\_len = 0, max\_ext\_diag\_data\_len = 0 and max\_address\_data\_len = 0. 244 bytes of input or output data can be achieved with max\_input\_data\_len = DP\_MAX\_INPUT\_DATA\_LEN or- max\_output\_data\_len = DP\_MAX\_OUTPUT\_DATA\_LEN.

### ident\_number

The ident number for a PROFIBUS DP device is assigned by the PNO.

DPS_DEFAULT_IDENT_NUMBER	The slave receives the ident number B205 <sub>hex</sub> (Softing DP slave)
0XXXXX	Any desired permissible ident number

### NOTE:

The ident number must correspond with the GSD file.

## user\_watchdog\_timeout

Application monitoring time for data access [ms]

0	Watchdog is switched off
1..65535	Watchdog timeout

## cfg\_data[DP\_MAX\_CFG\_DATA\_LEN] Initial configuration data

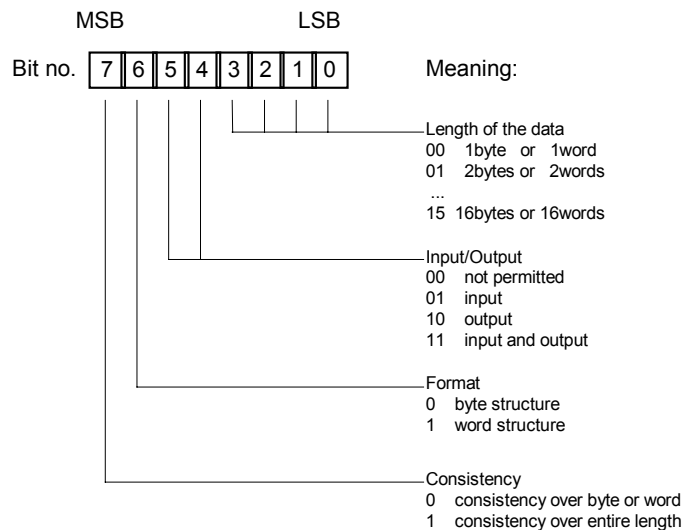
The configuration data define the ranges of the input and output areas and specify information on data consistency. The format complies with the form stipulated by the DP standard. Configuration data are coded differently according to use.

It is recommended to divide the input and/or output areas into logic or physical modules. The logic modules do not need to correspond with the physical modules. For each module you decide whether the configuration data are to be coded as an ID byte or as a special ID format. The corresponding bytes per module are arranged successively in any desired order. The order of the CFG bytes is used in ID-related and channel-related diagnostics as well as for the Data\_Exchange order.

There are two ways to represent the configuration data:

### ID byte

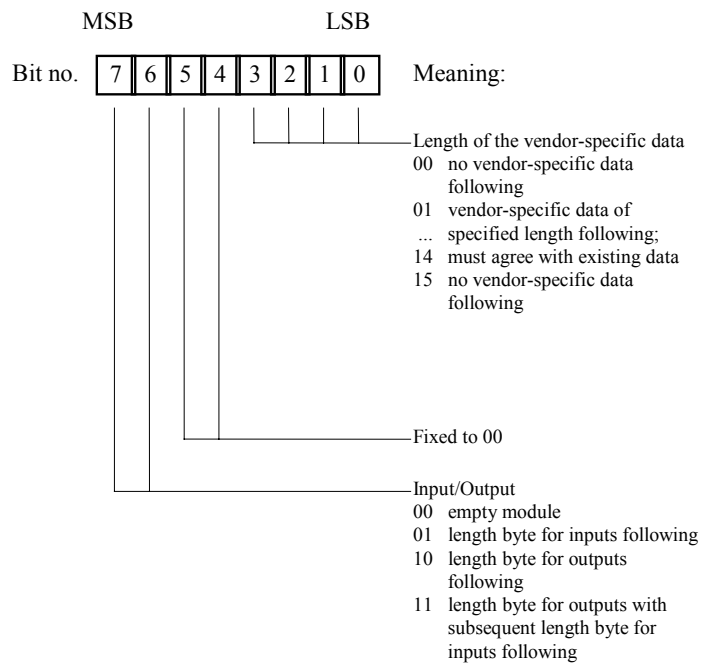
Use: Data length less than or equal to 16 words, no vendor-specific data



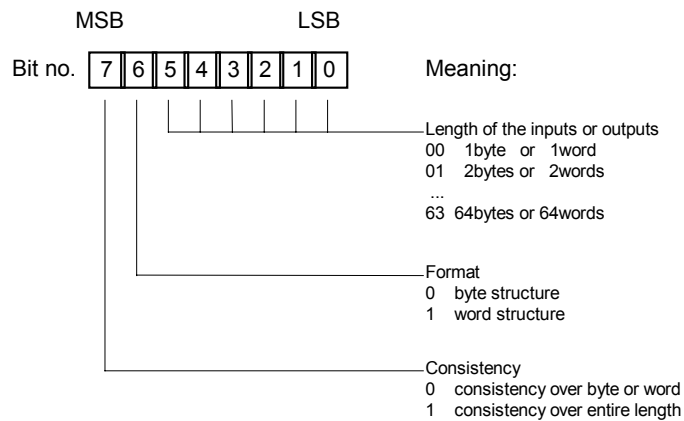


## Special ID format

Use: Data length less than or equal to 64 words, vendor-specific data



A length byte has the following structure:



Example of configuration data:

Module 1: 5 bytes input data, consistency over byte ID byte: 00010100 or 14<sub>hex</sub>

Module 2: 20 words output data, consistency over entire length, 3 bytes vendor-specific data  
Special ID format: 10000011 or 83<sub>hex</sub>  
Length byte: 11110101 or F5<sub>hex</sub>  
1st byte vendor-specific data  
2nd byte vendor-specific data  
3rd byte vendor-specific data

Module 3: 8 words input data, consistency over entire length  
ID byte: 11010111 or D7<sub>hex</sub>

Since you only need to pay attention to the order of the bytes within a module, `cfg_data` may have the following format:

```
cfg_data[0] = 14hex  
cfg_data[1] = D7hex  
cfg_data[2] = 83hex  
cfg_data[3] = F5hex  
cfg_data[4] = 1st byte vendor-specific data  
cfg_data[5] = 2nd byte vendor-specific data  
cfg_data[6] = 3rd byte vendor-specific data
```

**NOTE:**

**The slave will only work consistently if the corresponding bit of the configuration data is set (see also 'Dps\_Chk\_Cfg').**

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_INIT_SLAVE
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data Block for Confirmation:

result = POS:

Data structure	T_DPS_INIT_SLAVE_CON	
USIGN16	status	E_DPS_OK
INT16	remaining_frame_memory	Remaining memory for telegram buffer

result = NEG:

Data structure	T_DPS_INIT_SLAVE_CON	
USIGN16	status	E_DPS_IV, E_DPS_NO
INT16	remaining_frame_memory	Negative values indicate insufficient space

## status

E_DPS_OK	The slave was initialized successfully.
E_DPS_IV	The service request contains an invalid parameter. The high byte of status contains the error cause; possible causes are: E_DPS_WRONG_SLAVE_ADDRESS E_DPS_SSA_REQUIRED E_DPS_WRONG_CFG_LEN E_DPS_WRONG_INPUT_LEN E_DPS_WRONG_OUTPUT_LEN E_DPS_WRONG_PRM_LEN E_DPS_WRONG_DIAG_LEN E_DPS_WRONG_SSA_LEN E_DPS_WRONG_ENHANCED_INIT_LEN E_DPS_NOT_ENOUGH_FRAME_MEMORY E_DPS_ILLEGAL_CFG_DATA.
E_DPS_NO error	The service cannot be executed in this operating state. The high byte of status contains the cause E_DPS_DUPLICATED_SERVICE.

## NOTE:

The memory layout is not changed bitwise. The change depends on the buffer type, the internal alignment and their combinations, instead. The values for the remaining or missing memory capacity may therefore change erratically.

### 3.2 Dps\_Exit\_Slave

The slave no longer participates in bus traffic. After calling the service 'Dps\_Exit\_Slave', the slave can be reinitialized with 'Dps\_Init\_Slave'.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS
USIGN8	service	DPS_EXIT_SLAVE
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data block for Request:*

Data structure                      T\_DPS\_EXIT\_SLAVE\_REQ

No parameters

#### *Service-Description-Block for Confirmation:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_EXIT_SLAVE
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS

#### *Data block for Confirmation:*

result = POS:

Data structure                      T\_DPS\_EXIT\_SLAVE\_CON

USIGN16                      status                      E\_DPS\_OK

The slave no longer participates in bus traffic.

### 3.3 Dps\_Get\_Status

The service Dps\_Get\_Status delivers data on the current operating state, diagnostic state, assigned ident number, input and output areas, supported functions, address of the associated master, transfer rate of the slave and, as applicable, expected user entries. If a relevant status change occurs, the slave automatically reports this by means of an indication. The service can always be called after initialization by using Dps\_Init\_Slave.

#### *Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS
USIGN8	service	DPS_GET_STATUS
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block for Request:*

Data structure T\_DPS\_GET\_STATUS\_REQ

No parameters

#### *Service-Description-Block for Confirmation and Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_GET_STATUS
USIGN8	primitive	CON / IND
INT8	invoke_id	not used
INT16	result	POS / NEG

#### *Data Block for Confirmation and Indication:*

result = POS / NEG:

Data structure T\_DPS\_GET\_STATUS\_CON\_IND

USIGN16	status	E_DPS_OK, E_DPS_TO
USIGN8	slave_state	Operating state of the DP slave
USIGN8	diag_state	Diagnostic state of the DP slave
USIGN16	ident_number	Ident number
USIGN8	number_inputs	Number of inputs [byte]
USIGN8	number_outputs	Number of outputs [byte]
USIGN8	slave_add	Current address of the DP slave
USIGN8	non_volatile_slave_add	DP slave address stored in nonvolatile memory
USIGN8	master_add	Address of the DP master that parameterized the slave and carries out the cyclic data exchange
USIGN8	baud_rate	Transfer rate detected by the ASIC (only for information)
PB_BOOL	sync_enabled	Status of the outputs
PB_BOOL	freeze_enabled	Status of the inputs
PB_BOOL	clear_data	DP master in operating state CLEAR
PB_BOOL	cfg_await_response	Response to the configuration data
PB_BOOL	prm_await_response	Response to the parameterization data
PB_BOOL	await_startup_inputs	Setting the input data
USIGN8	reserved[16]	Reserved for enhancements. (0 =Ignore)

## status

E_DPS_OK	The status interrogation was executed correctly.
E_DPS_TO	The application monitoring time has run out. It can be reset by setting a new diagnostic which sets the bit DPS_DIAG_BIT_STAT_DIAG to 0.

## slave\_state

DPS_STATE_IDLE	The slave has not yet detected a transfer rate or no bus traffic is taking place.
DPS_STATE_WAIT_PRM	The slave could detect a valid transfer rate but has not yet received any parameterization data from the master
DPS_STATE_WAIT_CFG	Parameterizing the slave has been concluded and configuration data are awaited.
DPS_STATE_DATA_EXCHANGE	The slave was configured successfully and is exchanging user data or is in diagnostic mode.
DPS_STATE_CONTROLLER_ERROR	Fatal error in the ASIC.

## diag\_state

00H	A standard diagnostic (simple status message) is present.
DPS_DIAG_BIT_EXT_DIAG	The diagnostic is rated as an important message (such as an alarm).
DPS_DIAG_BIT_STAT_DIAG	The master will retrieve diagnostic information until this bit is cleared again. During this time no user data are exchanged although the slave is in DATA_EXCHANGE state.
DPS_DIAG_BIT_EXT_DIAG_OVERFLOW	This bit can be influenced by both master and slave: The master sets the bit when the slave transmits more diagnostic information than the master can take into account in its diagnostic buffer. The slave user sets it when there are more diagnostic information than the firmware can process; not even by means of repeated 'Dps_Slave_Diag.req'.

## ident\_number

The ident number currently used for the slave is displayed with this parameter.

## number\_inputs

0..244 bytes of input data; information on the consistency can only be determined from the configuration data.

## number\_outputs

0..244 bytes of output data; information on the consistency can only be determined from the configuration data.

## slave\_add

0..125

Current bus address of the slave

126

Default address of a slave that is being commissioned for the first time and receives a valid address through 'set\_slave\_add'. No data exchange takes place at this address.

## non\_volatile\_slave\_add

0..126

taken

An address that the master assigns to the slave with the service 'set\_slave\_add' is over into nonvolatile memory.

## master\_add

0..125

Master address

DP\_NO\_MASTER\_ADDRESS

The slave has not been assigned to a master.

## baud\_rate

DPS\_KBAUD\_9\_6

9.6 kbits/s

DPS\_KBAUD\_19\_2

19.2 kbits/s

DPS\_KBAUD\_45\_45

45.45kbits/s

DPS\_KBAUD\_93\_75

93.75 kbits/s

DPS\_KBAUD\_187\_5

187.5 kbits/s

DPS\_KBAUD\_500

500 kbits/s

DPS\_MBAUD\_1\_5

1.5 Mbits/s

DPS\_MBAUD\_3

3 Mbits/s

DPS\_MBAUD\_6

6 Mbits/s

DPS\_MBAUD\_12

12 Mbits/s

DPS\_NO\_BUS\_TRAFFIC

There is no bus traffic or the baud rate was not detected.

**sync\_enabled**

PB_TRUE	The slave supports the function for freezing the output data.
PB_FALSE	The output data cannot be frozen by using Global_Control.

**freeze\_enabled**

PB_TRUE	The input data can be frozen by using Global_Control.
PB_FALSE	The slave does not support the function for freezing the input data.

**clear\_data**

PB_TRUE failsafe	The master is in the operating state CLEAR; the output data are therefore set to state (to 0).
PB_FALSE	The master is not in CLEAR state but in a different operating state.

**cfg\_await\_response**

PB_TRUE 'Dps_Chk_Cfg'.	The user must approve or reject the configuration data with the service
PB_FALSE	No response expected from the user.

**prm\_await\_response**

PB_TRUE	An approval or a rejection of the parameterization data is being awaited. The service 'Dps_Set_Prm' is still to come.
PB_FALSE	No response expected from the user.

**await\_startup\_inputs**

PB_TRUE	The input data must be set by the user by means of 'profi_set_dps_input_data'. Otherwise, the slave will remain in the diagnostic state 'station_not_ready'.
PB_FALSE	The input data have been assigned a valid value by default.



### 3.4 Dps\_Slave\_Diag

The status or error information stored in the DP slave is referred to as diagnostic information. A diagnostic can be rated as an important message (such as an alarm) or a simple status message (such as resetting the alarm).

A diagnostic consists of standard and extended diagnostic information. The extended diagnostic information contains the status messages, the meaning of which must be declared according to the specific application, unless you are using one of the predefined types of diagnostic.

Due to the functionality of the firmware, the user can only set three diagnostic bits of the standard diagnostic to active, the remainder is filled in by the firmware.

*Service-Description-Block for Request:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS
USIGN8	service	DPS_SLAVE_DIAG
USIGN8	primitive	REQ
INT8	invoke_id	not used
INT16	result	not used

*Data Block for Request:*

Data structure	T_DPS_SLAVE_DIAG_REQ	
USIGN8	diag_state	Diagnostic state
USIGN8	ext_diag_data_len	Length of the slave-specific diagnostic data [byte] (0..DP_MAX_EXT_DIAG_DATA_LEN)
USIGN8	ext_diag_data[DP_MAX_EXT_DIAG_DATA_LEN]	Slave-specific diagnostic data

#### diag\_state

The individual bits of the diagnostic state result in the slave's diagnostic state to be set. This is performed through an OR operation.

00H	A standard diagnostic (simple status message) is present (corresponds to "OK").
DPS_DIAG_BIT_EXT_DIAG	The diagnostic is rated as an important message (such as an alarm), i.e. it must be treated with a high precedence.
DPS_DIAG_BIT_STAT_DIAG	"Static diagnostic": The master will retrieve diagnostic information until this bit is cleared again. During this time, no user data are exchanged.
DPS_DIAG_BIT_EXT_DIAG_OVERFLOW	If this bit is set, there are more diagnostic information than can be processed by means of Dps_Set_Slave_Diag.

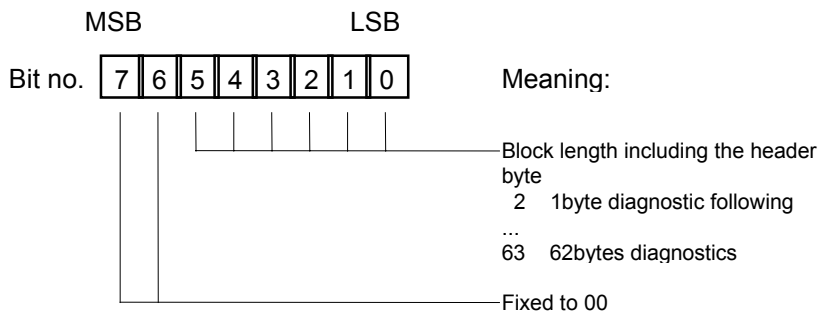
#### NOTE:

The bit 'DPS\_DIAG\_BIT\_EXT\_DIAG' does not indicate that ext\_diag\_data exist; it only shows the meaning of the diagnostic (i.e. alarm or status message).

## Device-related diagnostic

The block of the device-related diagnostic consists of a header byte and a variable number of bytes with device-specific diagnostic data. In this block, general diagnostic information, such as excess temperature or over- or undervoltage, is stored. Coding is determined depending on the specific device. Further evaluation requires the Ident\_Number and the GSD file.

Header byte:



Example of a device-related diagnostic:

Device-related diagnostic: 00000100 or 04<sub>hex</sub>

ext\_diag\_data[0] = 04<sub>hex</sub>

ext\_diag\_data[1] = device-specific diagnostic

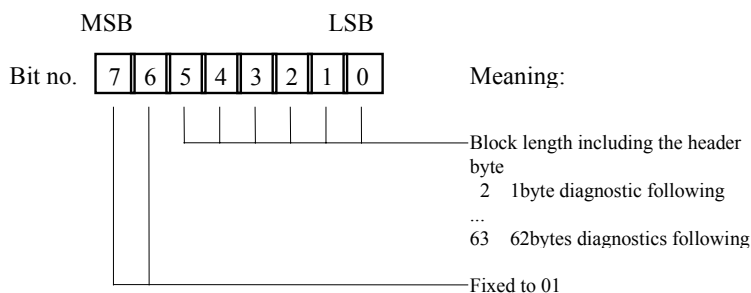
ext\_diag\_data[2] = device-specific diagnostic

ext\_diag\_data[3] = device-specific diagnostic

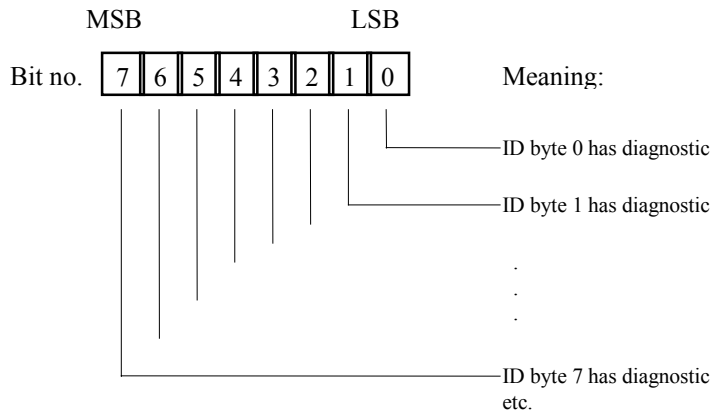
## ID-related diagnostic (module diagnostic)

The block of the ID-related diagnostic consists of a header byte and a variable number of bytes with ID-related diagnostic data. One bit is reserved for each ID byte assigned during configuration. The structure of each bit is filled in to the byte boundary. In this process, the value zero must be assigned to not configured bits. A set bit means that a diagnosis is pending in this I/O area but does not indicate the type of diagnostic. The order that was determined during configuration must be kept.

Header byte:



The bit structure for the ID-related diagnostic is as follows:



Example of an ID-related diagnostic:

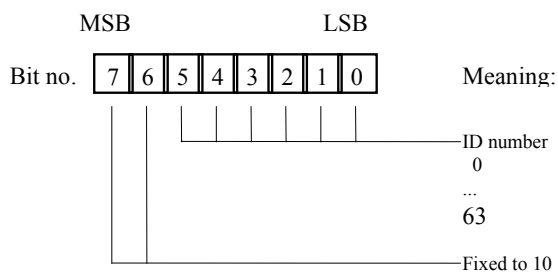
ID-related diagnostic: 01000101 or 45<sub>hex</sub>  
 ID number 0 with diagnostic: 00000001 or 01<sub>hex</sub>  
 ID number 12 with diagnostic: 00010000 or 10<sub>hex</sub>  
 ID number 18 with diagnostic: 00000100 or 04<sub>hex</sub>

ext\_diag\_data[0] = 45<sub>hex</sub>  
 ext\_diag\_data[1] = 01<sub>hex</sub>  
 ext\_diag\_data[2] = 10<sub>hex</sub>  
 ext\_diag\_data[3] = 04<sub>hex</sub>  
 ext\_diag\_data[4] = 00<sub>hex</sub>

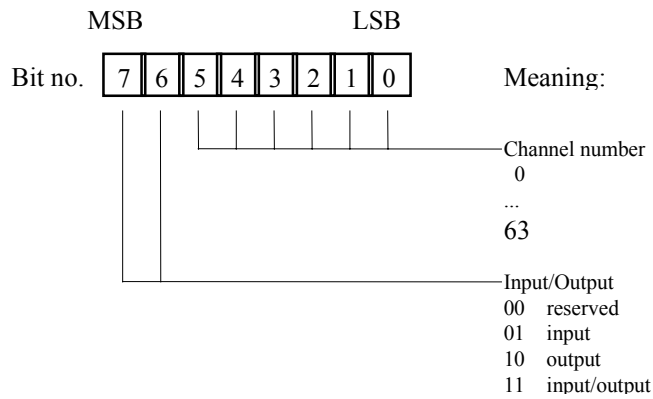
## Channel-related diagnostic

In this block, the diagnosed channels and the diagnostic causes are entered successively. The length per entry is three bytes.

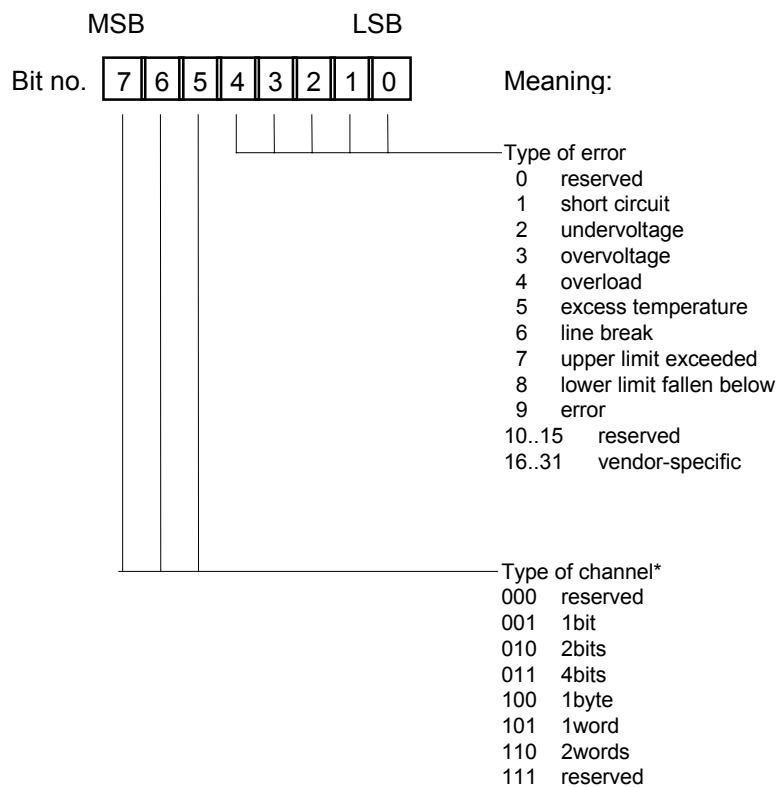
**ID number:**



### Channel number:



### Type of diagnostic:



**Note:** The channel type is used for diagnostics only and has an influence on the IDs and modules, respectively, that were determined in the

Example of a channel-related diagnostic:

Channel-related diagnostic with ID number 0:	10000000	or	80 <sub>hex</sub>
Channel2 as input:	01000010	or	42 <sub>hex</sub>
Overload, channel organized bitwise:	00100100	or	24 <sub>hex</sub>
Channel-related diagnostic with ID number 12:	10001100	or	8C <sub>hex</sub>
Channel6 as output:	10000110	or	86 <sub>hex</sub>
Upper limit exceeded, channel organized wordwise:	10100111	or	A7 <sub>hex</sub>

```
ext_diag_data[0] = 80hex
ext_diag_data[1] = 42hex
ext_diag_data[2] = 24hex
ext_diag_data[3] = 8Chex
ext_diag_data[4] = 86hex
ext_diag_data[5] = A7hex
```

## Service-Description-Block for Confirmation:

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_SLAVE_DIAG
USIGN8	primitive	CON
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data Block for Confirmation:

result = POS:

Data structure	T_DPS_SLAVE_DIAG_CON	
USIGN16	status	E_DPS_OK

result = NEG:

Data structure	T_DPS_SLAVE_DIAG_CON	
USIGN16	status	E_DPS_IV, E_DPS_NO

## status

E_DPS_OK	The diagnostic data were set.
E_DPS_IV	The service request contains an invalid parameter. The high byte of status contains the error cause; possible causes are: E_DPS_WRONG_DIAG_LEN E_DPS_INVALID_DIAG_STATE E_DPS_INVALID_EXT_DIAG_DATA.
E_DPS_NO	The service cannot be executed in this operating state (the service 'Dps_Init_Slave' was not executed).

### 3.5 Dps\_Chk\_Cfg

The configuration data describe the size of the input and output data areas and provide information on their data consistency. The format is defined in the DP standard (see also 'Dps\_Init\_Slave').

The configuration data transmitted by the master are automatically compared with the actual configuration of the slave if the parameter 'auto\_cfg\_response' was set to PB\_TRUE during the initialization of the slave. The configuration is accepted if the format and the length specifications as well as the input/output areas agree. A consistency is only assured if the applicable bit of the configuration data has been set. Consistency means, with regard to the slave, that the data are always read and written completely. Should all data require no consistency, the slave will copy bitwise without synchronization, a process which takes a bit less time. The interrupt mode of the PAPI always works consistently.

The user will only be informed of new configuration data if the parameter 'auto\_cfg\_response' was set to PB\_FALSE during the initialization of the slave. The indication passes on the nominal configuration to the user for checking. With the response, the user decides whether the configuration is approved or rejected. Whether a response is still to come is indicated by the parameter 'cfg\_wait\_response' of the service 'Dps\_Get\_Status'.

#### *Service-Description-Block of the Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_CHK_CFG
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	POS

#### *Data Block of the Indication:*

Data structure	T_DPS_CHK_CFG_IND	
USIGN16	status	E_DPS_OK: Configuration data were received
USIGN8	cfg_data_len	Length of the configuration data (1..DP_MAX_CFG_DATA_LEN)
PB_BOOL	cfg_await_response	Response to the indication PB_TRUE: A response is being awaited (the user still needs to generate a Dps_Chk_Cfg.res)
PB_BOOL	cfg_await_response	Response to the indication

## Service-Description-Block of the Response:

USIGN16	comm_ref	not used
USIGN8	layer	DPS
USIGN8	service	DPS_CHK_CFG
USIGN8	primitive	RES
INT8	invoke_id	not used
INT16	result	POS / NEG

## Data Block of the Response :

result = POS:

Data structure	T_DPS_CHK_CFG_RES	
USIGN16	status	E_DPS_OK

result = NEG:

Data structure	T_DPS_CHK_CFG_RES	
USIGN16	status	E_DPS_NO

## status

E_DPS_OK	The configuration is approved.
E_DPS_NO	The configuration is rejected and the slave changes into WAIT_PRM state.

### 3.6 Dps\_Set\_Prm

The parameterization of the DP slave is initially performed in the startup and runup phases of the DP system, but is also permitted in user data exchange.

The user will be informed of new parameterization data provided that user\_prm\_data are contained, independent of the initialization of the slave. The parameter 'auto\_prm\_response' determines whether the parameterization data are to be transferred to the user for checking and require a response.

#### *Service-Description-Block of the Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_SET_PRM
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	not used

#### *Data Block of the Indication:*

Data structure	T_DPS_SET_PRM_IND	
USIGN16	status	E_DPS_OK: Parameterization data were received
USIGN8	user_prm_data_len	Length of the vendor-specific parameterization data (0..DP_MAX_USER_PRM_DATA_LEN)
USIGN8	prm_await_response	Response to the indication
USIGN8	station_status	Status of the DP slave
USIGN8	wd_fact_1	Factors of response monitoring
USIGN8	wd_fact_2	
USIGN8	min_tsd	Minimum reaction time [T <sub>bit</sub> ]
USIGN16	ident_number	The ident number that is currently used for the slave is displayed with this parameter.
USIGN8	group_ident	Identification bits for the formation of groups in Global_Control.
USIGN8	user_prm_data [DP_MAX_USER_PRM_DATA_LEN]	Vendor-specific parameterization data (such as limits or controller parameters)

#### **prm\_await\_response**

PB_TRUE	The parameterization data must be checked by the user and a corresponding response must be generated.
PB_FALSE	The indication only serves for reporting vendor-specific parameterization data, if there are any.
A	response by the user is not required.



## station\_status

DP_PRM_WD_ON	The response monitoring of the slave has been enabled. It assures that, in the case of a failure of the master, the outputs are switched into failsafe state (CLEAR mode) as soon as the response monitoring time runs out.
DP_PRM_FREEZE_REQ	The slave is to be operated in Freeze mode as soon as it receives a corresponding command (Global_Control).
DP_PRM_SYNC_REQ	The slave is to be operated in Sync mode as soon as it receives a corresponding Global_Control command.
DP_PRM_UNLOCK_REQ	
DP_PRM_LOCK_REQ	For the meaning please refer to the table. These parameters cannot be influenced by the user; they are evaluated by the firmware and are only for information.

DP_PRM_LOCK_REQ	DP_PRM_RM_UNLOCK_REQ	Meaning
0	0	min TSDR is overwritten, all other parameters remain unaffected
0	1	DP slave is enabled for other masters
1	0	DP slave is disabled for other masters, all other parameters are taken over (exception: min TSDR = 0)
1	1	DP slave is enabled for other masters

**Table 6-1:** Determination of the bits DP\_PRM\_LOCK\_REQ and DP\_PRM\_UNLOCK\_REQ

## wd\_fact\_1, wd\_fact\_2

From the above factors, the response monitoring time can be calculated as follows:

- Response monitoring time [ms] = Time basis of the watchdog \* wd\_fact\_1 \* wd\_fact\_2

The time basis of the watchdog can be fixed to 1ms or 10ms (default: 10ms). It is determined in the first byte of the vendor-specific parameterization data (see parameter 'user\_prm\_data'). 0..255

## NOTE:

**It is not recommended to use a response monitoring time of less than 2ms or 20ms since the times are below a timer tick and can therefore run out immediately. Monitoring times between 2ms or 20ms and 650s can be realized reliably.**

## min\_tsdr

The minimum reaction time is the length of time that the slave must at least wait before it is permitted to send its reply telegrams back to the DP master.

0.. 10	The previous value for the minimum reaction time is retained
11..255	Minimum reaction time

## group\_ident

The identification bit is assigned by the master during the runup phase. A slave may belong to several groups. Each identification bit represents a group. The slave is addressed whenever the bit of its group is set in the Global\_Control command.

## user\_prm\_data [DP\_MAX\_USER\_PRM\_DATA\_LEN]:

### NOTE:

The first byte of the vendor-specific data cannot be used freely since it is preassigned by the ASIC. The individual bits have the following meanings:

Bit7:	Fixed to 0
Bit6:	Fixed to 0
Bit5:	Fixed to 0
Bit4:	Fixed to 0
Bit3:	Fixed to 0
Bit2 (DPS_SPC3_USR_PRM_WD_BASE_1MS):	0 ⇒ Time basis of watchdog is 10ms 1 ⇒ Time basis of watchdog is 1ms
Bit1 (DPS_SPC3_USR_PRM_DISABLE_STOPBIT):	0 ⇒ Stop bit monitoring is not switched off 1 ⇒ Stop bit monitoring is switched off
Bit0 (DPS_SPC3_USR_PRM_DISABLE_STARTBIT):	0 ⇒ Start bit monitoring is not switched off 1 ⇒ Start bit monitoring is switched off.

## *Service-Description-Block of the Response:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS
USIGN8	service	DPS_SET_PRM
USIGN8	primitive	RES
INT8	invoke_id	not used
INT16	result	POS / NEG

## *Data Block of the Response:*

result = POS:

Data structure	T_DPS_SET_PRM_RES	
USIGN16	status	E_DPS_OK

result = NEG:

Data structure	T_DPS_SET_PRM_RES	
USIGN16	status	E_DPS_NO

## **status**

E_DPS_OK	The parameterization data are approved.
E_DPS_NO	The parameterization data are rejected, the slave remains in WAIT_PRM state.

### 3.7 Dps\_Set\_Slave\_Add

If during initialization the parameter 'set\_slave\_add\_supported' was set, the DP slave supports an address assignment over the bus. When the slave is assigned a new address, this is reported by means of an indication. The slave is then in the operating state WAIT\_PRM. The new address is stored to the NVRAM.

This service is only provided for informing the user and cannot be executed actively (you cannot transmit a request). It is determined with the help of Dps\_Init\_Slave.

*Service-Description-Block of the Indication:*

USIGN16	comm_ref	not used
USIGN8	layer	DPS_USR
USIGN8	service	DPS_SET_SLAVE_ADD
USIGN8	primitive	IND
INT8	invoke_id	not used
INT16	result	not used

*Data Block of the Indication:*

Data structure	T_DPS_SET_SLAVE_ADD_IND	
USIGN16	status	E_DPS_OK: The slave address was changed and the new address stored in nonvolatile memory.
USIGN8	rem_slave_data_len	Length of the user-defined data (0..DP_MAX_REM_SLAVE_DATA_LEN)
USIGN8	new_slave_add	New slave address
USIGN16	ident_number	Ident number assigned by the PTO.
PB_BOOL	no_add_chg	Changing the slave address
OCTET	rem_slave_data [DP_MAX_REM_SLAVE_DATA_LEN]	User-defined data

#### new\_slave\_add

0..125      The new address was stored in nonvolatile memory and can be reused with the service 'Dps\_Init\_Slave' via NON\_VOLATILE\_SLAVE\_ADD.

#### no\_add\_chg

This parameter is not stored to the NVRAM. It therefore only remains effective until the slave is restarted.

PB_TRUE	The slave address can be changed once again later.
PB_FALSE	The station address of the slave can no longer be changed over the bus until the slave is restarted.

**NOTE:**

The master cannot determine for sure whether the address assignment was successful. You should therefore check the address assignment with the help of a slave diagnostic.

The master can only change the slave address if the slave is currently not in cyclic data exchange with the master.



# **PROFIBUS Application Program Interface**

## **Tools Library**

Version 5.2  
Rev. 01

Date: 03-March-1998

Softing AG  
Richard-Reitzner-Allee 6  
D-85540 Haar  
Phone (++49) 89 45 65 6 - 0  
Fax (++49) 89 45 65 6 - 399

© Copyright by Softing AG, 1989-2003  
All rights reserved.

## **Copyright Notice**

All rights are reserved. No part of these instructions may be reproduced (printed material, photocopies, microfilm or other method) or processed, copied or distributed using electronic systems in any form whatsoever without prior written permission of Softing AG.

The producer reserves the right to make changes to the scope of supply as well as changes to technical data, even without prior notice.

A great deal of attention was made to the quality and functional integrity in designing, manufacturing and testing the system. However, no liability can be assumed for potential errors that might exist or for their effects. Should you find errors please inform your distributor of the nature of the errors and the circumstances under which they occur. We will be responsive to all reasonable ideas and will follow up on them, taking measures to improve the product if necessary.

We call your attention to the fact that the company name and trademark as well as product names are, as a rule, protected by trademark, patent and product brand laws.

Copyright 1989-2003 by Softing AG, Haar

---



CONTENTS

1 SCOPE ..... 1

2 BUS PARAMETER SETS ..... 2

    2.1 GET DEFAULT BUS PARAMETERS ..... 2

    2.2 TABLES OF DEFAULT BUS PARAMETER SETS..... 3

        2.2.1 Recommended Bus Parameters for FMS/FM7 Operation using ASPC2 ..... 3

        2.2.2 Recommended Bus Parameters for simultaneous DP / FMS Operation ..... 4

        2.2.3 Recommended Bus Parameters for DP Operation ..... 5

3 COMMUNICATION RELATIONSHIP LIST RESOURCES ..... 6

    3.1 RESOURCES INIT ..... 6

    3.2 RESOURCES ADD ENTRY ..... 7



### 1 SCOPE

This manual describes the User Toolkit for easy configuration of SOFTING's PROFIBUS controllers.

The Toolkit provides C-functions to

- Get default bus parameter sets
- Calculate CRL memory requirements.

## 2 BUS PARAMETER SETS

Setting up a consistent set of bus parameters for a PROFIBUS network is no trivial task. For best performance the parameter set has to be calculated individually, among others regarding the number of masters and slaves, as well as the amount of data to be transferred in a cycle. Nevertheless default sets of parameters that will work with most PROFIBUS configurations can be given for all baud rates (see also manual Basic Management chapter 3.2.1).

### 2.1 GET DEFAULT BUS PARAMETERS

The function *pbt\_get\_fmb\_def\_bus\_param* provides default bus parameters for PROFIBUS stations dependent of the desired baud\_rate and operation mode.

The function has the following prototype:

```
extern PB_BOOL pbt_get_fmb_def_bus_param
(
    IN    USIGN8          baud_rate,
    IN    USIGN8          station_addr,
    IN    PB_BOOL         in_ring_desired,
    IN    UNSIGN16        mode
    OUT   T_FMB_SET_BUSPARAMETER_REQ FAR* bus_param_ptr
);
```

Function parameter description:

baud_rate:	desired baudrate	(see valid baudrates)
station_addr:	desired station address	(0..126)
in_ring_desired:	PB_TRUE ⇒	active station (master)
	PB_FALSE ⇒	passive station (slave)
mode:	DP_MODE ⇒	standalone DP operation
	DP_FMS_FM7_MODE ⇒	simultaneous DP/FMS/FM7 operation
	FMS_FM7_MODE ⇒	standalone FMS/FM7 operation
bus_param_ptr:	pointer to FMB bus parameter structure	

#### Possible function return values:

- PB_TRUE	parameters correctly set
- PB_FALSE	no parameter set available, no parameters set

## 2.2 TABLES OF DEFAULT BUS PARAMETER SETS

This chapter of the manual describes the **Default Bus Parameter Sets** up to 12000 kbit/s that are returned by the function `pbt_get_fmb_def_bus_param`.

The values are taken from EN 50170/2 (FDL and DP) and from the implementation guide to EN E 50170 / 2 (DP) recommendations for ASIC ASPC2 based hardware platforms.

The following tables show the settings for the *Default Bus Parameter Sets*:

### 2.2.1 Recommended Bus Parameters for FMS/FM7 Operation using ASPC2

Parameter/Baudrate	9,6 Kbaud	19,2 Kbaud	45,45 Kbaud	93,75 Kbaud	187,5 KBaud	500 KBaud	1,5 MBaud	3 MBaud	6 MBaud	12 MBaud
T <sub>SL</sub> [bit times]	100	200	-	500	1000	2000	3000	400	600	1000
min_T <sub>SDR</sub> [bit times]	30	60	-	125	250	500	150	11	11	11
max_T <sub>SDR</sub> [bit times]	50	100	-	250	500	1000	980	250	450	800
T <sub>SET</sub> [bit times]	5	10	-	15	25	50	240	4	8	16
T <sub>QUI</sub> [bit times]	22	22	-	22	22	22	0	3	6	9
G	1	1	-	1	1	1	10	10	10	10
HSA	126	126	-	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	-	1	1	1	1	2	3	4
T <sub>TR</sub> [bit times]	10000	15000	-	30000	50000	100000	300000	600000	1200000	2400000

The default values for 45,45 kbit/s are not defined for single FMS operation.

## 2.2.2 Recommended Bus Parameters for simultaneous DP / FMS Operation

Parameter/Baudrate	9,6 Kbaud	19,2 Kbaud	45,45 Kbaud	93,75 Kbaud	187,5 Kbaud	500 Kbaud	1,5 Mbaud	3 Mbaud	6 Mbaud	12 Mbaud
T <sub>SL</sub> [bit times]	125	250	640	600	1500	3500	3000	400	600	1000
min_T <sub>SDR</sub> [bit times]	30	60	11	125	250	500	150	11	11	11
max_T <sub>SDR</sub> [bit times]	60	120	400	250	500	1000	980	250	450	800
T <sub>SET</sub> [bit times]	1	1	95	1	1	1	240	4	8	16
T <sub>QUI</sub> [bit times]	0	0	0	0	0	0	0	3	6	9
G	1	1	10	1	1	1	10	10	10	10
HSA	126	126	126	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	1	1	1	1	1	2	3	4
T <sub>TR</sub> [Bitzeiten]	-	-	-	-	-	-	-	-	-	-

The baudrate 45,45 kbit/s is only used for DP- and PA -systems with coupling devices.

### 2.2.3 Recommended Bus Parameters for DP Operation

Parameter/Baudrate	9,6 Kbaud	19,2 Kbaud	45,45 Kbaud	93,75 Kbaud	187,5 Kbaud	500 Kbaud	1,5 Mbaud	3 Mbaud	6 Mbaud	12 Mbaud
T <sub>SL</sub> [bit times]	100	100	640	100	100	200	300	400	600	1000
min_T <sub>SDR</sub> [bit times]	11	11	11	11	11	11	11	11	11	11
max_T <sub>SDR</sub> [bit times]	60	60	400	60	60	100	150	250	450	800
T <sub>SET</sub> [bit times]	1	1	95	1	1	1	1	4	8	16
T <sub>QUI</sub> [bit times]	0	0	0	0	0	0	0	3	6	9
G	1	1	10	1	1	1	10	10	10	10
HSA	126	126	126	126	126	126	126	126	126	126
Max_Retry_Limit	1	1	1	1	1	1	1	2	3	4
T <sub>TR</sub> [Bitzeiten]	-	-	-	-	-	-	-	-	-	-

The baudrate 45,45 kbit/s is only used for DP- and PA -systems with coupling devices.

### 3 COMMUNICATION RELATIONSHIP LIST RESOURCES

The memory requirements of a CRL depend on the number of CRL entries, the connection types, the number of parallel services, etc. (see also manual Basic Management chapter 4.1.2).

A comfortable way to find out the memory requirements is offered by the following functions. These functions do low level checks on a CRL and calculate the memory requirements.

#### 3.1 RESOURCES INIT

At first the function *ccrl\_resrcs\_init* has to be called to initialize the internal structures for calculating the memory requirements.

The function has the following prototype:

```
extern VOID ccrl_resrcs_init
(
    IN      T_FM7_CRL_HDR    FAR*   crl_hdr
);
```

Function parameter description:

crl\_hdr:            pointer to CRL header structure

**Possible function return values:**

- NONE



### 3.2 RESOURCES ADD ENTRY

For each CRL entry the function *ccrl\_resrces\_add\_entry* has to be called to evaluate the number of resources.

Two actions are performed by this function. First, it checks the CRL entry. If the entry is incorrect it tries to correct it. If the entry cannot be corrected it returns with a negative result. If the entry is OK, *ccrl\_resrces\_add\_entry* calculates resources. After function return, the output buffer holds the sum of resources that are needed by all CRL entries that were put into *ccrl\_resrces\_add\_entry* since last call of *ccrl\_resrces\_init*. The result after the last call of *ccrl\_resrces\_add\_entry* this is the number of all needed resources for the whole CRL. This result can be used as input for the PROFIBUS configuration service.

The function has the following prototype:

```
extern USIGN16 ccrl_resrces_add_entry
(
    IN      USIGN16          cr,
    IN      T_FM7_CRL_STATIC FAR*  crl_ptr,
    OUT     T_FMB_CONFIG_CRL FAR*  config_ptr
);
```

Function parameter description:

cr: communication reference  
crl\_ptr: pointer to static part of the CRL entry  
config\_ptr: pointer to CRL configuration structure

#### Possible function return values:

- E_OK	function executed correctly
- E_FM7_CRL_INVALID_ENTRY	invalid entry found in CRL, resource not calculated