Protocol API

PROFIBUS DP-Master

V2.8.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC061001API22EN | Revision 22 | English | 2017-09 | Released | Public

Table of Contents

1 Introduction 5
1.1 About this Document 5
1.2 List of Revisions 5
1.3 Functional Overview 6
1.4 System Requirements 6
1.5 Intended Audience 6
1.6 Specifications 7
1.6.1 Technical Data 7
1.7 Terms, Abbreviations and Definitions 9
1.8 References to Documents 9

2 Getting started 10
2.1 Overview 10
2.2 Task Structure 10

3 PROFIBUS Functions 13
3.1 PROFIBUS DP and the OSI/ISO Layer Model 13
3.2 PROFIBUS DP Operation Modes (States) 14
3.3 Functionality of the FSPMM-Task (Layer 7) 15
3.3.1 Cyclic Data Transfer 15
3.3.2 Acyclic Data Transfer 17
3.3.3 Configuration of Inputs and Outputs at Slaves 19
3.3.4 Alarm Processing 24
3.3.5 Diagnosis 26
3.4 Functionality of the FSPMM2-Task 31
3.4.1 Overview of Supported Functionality 31
3.4.2 General Remarks on DP V1 Class2 31
3.4.3 DP V1 Class2 Masters 32
3.4.4 Basic Services for Connection Maintenance 32
3.4.5 Basic Services available after Establishing a DP V1-Class2 Connection 33
3.4.6 Extended Addressing Mechanism 33
3.4.7 Short Description of the MSAC2M State Machine 34
3.4.8 Initialization Process of a DP V1-Class2 Connection (MSAC2_Initiate) 35
3.4.9 Detailed Description of DP V1 Class2 Initialization Parameters 37
3.4.10 Execution of Basic Services 39
3.4.11 Requirements for the Class2 Master's User Task 42
3.4.12 Connection Supervision by Internal Timers 42
3.5 Functionality of the DL Task (Layer 2) 43
3.5.1 Data Transfer Services 44
3.5.2 Management of Services Access Points 46
3.5.3 Management of Layer 2 System Variables and Tasks 48
3.6 Configuration during running system (Operation State OPERATE) 49
3.6.1 Configuration Procedure 49
3.7 Redundancy Functionality 52

4 Configuration of Bus and Slave Parameters 54
4.1 Configuring the Master using Packets 54
4.2 Detailed Description of Bus and Master Parameters 55
4.3 Detailed Description of Slave Parameters 62

5 Status Information 70
5.1 Common Status 70
5.2 Extended Status Block 70
5.2.1 Extended Status PROFIBUS DP Master 71
5.2.2 Extended Status Field PROFIBUS DP Master 76

6 The Application Interface 78
6.1 The FSPMM-Task 79
6.1.1 PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register 81
6.1.2 PROFIBUS_FSPMM_CMD_INIT_REQ/CNF – Initialization Command 83
6.1.3 PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode 86

6.1.4 PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication 89
6.1.5 PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic..... 91
6.1.6 PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic..... 95
6.1.7 PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output Data 96
6.1.8 PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input Data 98
6.1.9 PROFIBUS_FSPMM_CMD_READ_REQ/CNF – DP V1 Class1 Read Request 101
6.1.10 PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – DP V1 Class1 Write Request..... 105
6.1.11 PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification 110
6.1.12 PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge 113
6.1.13 PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ/CNF – Download Slave Parameter Set 118
6.1.14 PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF – Global Control Message 125
6.1.15 PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ/CNF – Activate/Deactivate Slaves..... 129
6.1.16 PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a Fatal Fault 131
6.1.17 PROFIBUS_MSCS1M_CMD_INIT_CS_REQ/CNF – Initialize ClockSync Feature..... 132
6.1.18 PROFIBUS_MSCS1M_CMD_SET_TIME_REQ – Set Time for *TimeEvent ClockValue* Sequence134
6.1.19 PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_REQ/CNF – Switch Redundancy..... 137
6.1.20 PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_REQ/CNF - Input Change of Slaves..... 140
6.1.21 PROFIBUS_FSPMM_CMD_UPLOAD_REQ/CNF – Upload Slave Parameter Set 142
6.2 The FSPMM2-Task..... 145
6.2.1 PROFIBUS_FSPMM2_CMD_INIT_REQ/CNF – Initialization Command 147
6.2.2 PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate DPV1 Class2 Connection..... 149
6.2.3 PROFIBUS_FSPMM2_CMD_READ_REQ/CNF – DP V1 Class2 Read Request 157
6.2.4 PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class2 Write Request 162
6.2.5 PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request 167
6.2.6 PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection 172
6.2.7 PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ/CNF – Read Slave Diagnostics (DP V1 Class2) 176
6.2.8 PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ/CNF – Read Input Values 178
6.2.9 PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ/CNF – Read Output Values 180
6.2.10 PROFIBUS_FSPMM2_CMD_GET_CFG_REQ/CNF – Get Configuration Command..... 182
6.2.11 PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ/CNF – Set Slave Address (DP V1 Class2)184
6.2.12 PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ/CNF – Get Master Diagnosis..... 187
6.2.13 PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ/CNF – Live List Command 189
6.2.14 PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication 192
6.2.15 PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication/Response 195
6.2.16 PROFIBUS_FSPMM2_CMD_RESET_CONN_REQ/CNF – Reset Connection Command..... 197
6.3 The DL-Task..... 199
6.3.1 PROFIBUS_DL_CMD_START_DLE_REQ/CNF – Starts the DL-Layer..... 200
6.3.2 PROFIBUS_DL_CMD_STOP_DLE_REQ/CNF – Stop DL Layer..... 202
6.3.3 PROFIBUS_DL_CMD_DATA_ACK_REQ/CNF - Send SDA Service..... 204
6.3.4 PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service 209
6.3.5 PROFIBUS_DL_CMD_DATA_REQ/CNF - Send SDN Service..... 211
6.3.6 PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication..... 215
6.3.7 PROFIBUS_DL_CMD_DATA_REPLY_REQ/CNF - Send SRD Service 217
6.3.8 PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication 221
6.3.9 PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ/CNF - Reply Update Service..... 224
6.3.10 PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request..... 228
6.3.11 PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ/CNF - Activate an SAP for Responder Functionality 233
6.3.12 PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request..... 238
6.3.13 PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set..... 240
6.3.14 PROFIBUS_DL_CMD_SET_VALUE_REQ/CNF - Set Value Service 244
6.3.15 PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ/CNF – Obtain FDL Status 247
6.3.16 PROFIBUS_DL_CMD_GET_LMS_REQ/CNF - Get List of active Stations..... 249
6.3.17 PROFIBUS_DL_CMD_REGISTER_LMS_REQ/CNF - Register an Application to receive LMS Change Indications 252
6.3.18 PROFIBUS_DL_CMD_LMS_CHANGED_IND - Indication for Change in LMS..... 255
6.3.19 PROFIBUS_DL_CMD_GET_LL_REQ/CNF – Get the Life List (List of Active or Passive Stations)257
6.4 The APM-Task..... 260
6.4.1 PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ/CNF – Set Master to active/passive 261
6.4.2 PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master264
6.5 Packets for Configuration during running system..... 265
6.5.1 RCX_VERIFY_DATABASE_REQ/CNF – Verify the new Configuration Database..... 266
6.5.2 RCX_ACTIVATE_DATABASE_REQ/CNF – Activate the new Configuration Database 269
6.6 Handshake Mode..... 271
6.6.1 Buffered / Input and Output Host-Controlled Mode 272

6.6.2 Bus-synchronous / Input and Output Host-Controlled Mode 273
6.6.3 Configuration 274
6.7 Network Scan 275

7 Topics for Linkable Object Module Users 276
7.1 Accessing the Protocol Stack by Programming the AP Task's Queue 276

8 Status/Error Codes Overview 277
8.1 Common Error Codes 277
8.2 Error Codes of the FSPMM-Task 280
8.2.1 Diagnostic Codes of the FSPMM-Task 282
8.3 Error Codes of the FSPMM2-Task 283
8.3.1 Diagnostic Codes of the FSPMM2-Task 283
8.4 Error Codes of the DL-Task 284
8.4.1 Diagnostic Codes of the DL-Task 285
8.5 Error Codes of the APM-Task 286
8.5.1 Diagnostic Codes of the APM-Task 287

9 Appendix 288
9.1 Legal Notes 288
9.2 List of Figures 292
9.3 List of Tables 292
9.4 Contacts 296

1 Introduction

1.1 About this Document

This manual describes the application interface of the PROFIBUS DP Master stack, with the aim to support and lead you during the integration process of the given stack into your own application.

Base of the development of the stack itself is the Hilscher's Task Layer Reference Programming Model. It is a description of how to program a task in general, which is defined as a combination of appropriate functions belonging to the same type of protocol layer. It furthermore defines of how different tasks have to communicate with each other in order to exchange their layer information in between. The reference model is commonly used by all programmers at Hilscher and shall be used by you as well when writing your application task on top of the stack.

1.2 List of Revisions

Rev	Date	Name	Revisions
21	2016-03-23	HH	Firmware / Stack version 2.7.0 Section *Detailed Description of Slave Parameters*: Description of Bus_Para_Len updated. Section *PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate DPV1 Class2 Connection* and *PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request*: Explanation for typical errors added.
22	2017-09-25	HH/RG/RH	Firmware / Stack version 2.8.0 Section *Handshake Mode* added. Section *Network Scan* added. General sections *Fundamentals* and *Dual-Port Memory* removed. Section *Status Information* added. Section *Topics for Linkable Object* Module Users added. Section *PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification* revised. Table 67 corrected.

Table 1: List of Revisions

1.3 Functional Overview

The stack has been designed in order to meet the IEC 61158 Type 3 specification. You as a user are getting a capable and a general-purpose Software package with following features:

- Implementation of the DP-standard cyclic state machine
- Implementation of the DP V1-extended acyclic commands read/write Class1 and alarm Class1.
- Baud rate ranging from 9.6 kBaud up to 12 MBaud

1.4 System Requirements

The software package has the following system requirements to its environment:

- netX-Chip as CPU hardware platform
- Operating system for task scheduling required
- Operating system independency, rcX or Windows CE are implemented as a reference
- Stack portable to any other processor technology.

1.5 Intended Audience

This manual is suitable for software developers with the following background:

- Knowledge of the programming language C
- Knowledge of the use of the real-time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model

Knowledge of the IEC 61158 specification is helpful.

1.6 Specifications

The data below applies to PROFIBUS DP Master firmware and stack version V2.8.0.

1.6.1 Technical Data

Supported State Machines

FSPMM – Field bus Service Protocol Master state machine

MSCY1M – Master to Slave cyclic state machine

DMPMM – Data Link Mapping Protocol Master state machine

MSAL1M – Master Class1 to Slave alarm state machine

MSAC1M – Master Class1 to Slave acyclic state machine

MMAC1 – Master to Master acyclic Class1 state machine

MSCS1M – Master Class3 to Slave clock sync state machine

Technical Data

Maximum number of supported slaves	125 (DPV0/DPV1)
Maximum number of total cyclic input data	5712 bytes (Status information is separately managed)
Maximum number of total cyclic output data	5760 bytes
Maximum number of cyclic input data	244 bytes/slave
Maximum number of cyclic output data	244 bytes/slave
Configuration data	max. 244 bytes per slave
Parameterization data per slave	7 bytes/slave standard parameters 237 bytes/slave application specific parameters
Acyclic communication	DPV1 Class1 Read/Write DPV1 Class1 Alarm DPV1 Class2 Initiate/Read/Write/Data transport/Abort
Maximum number of acyclic read/write	240 bytes per slave and telegram
Functions	Configuration in Run (Requires host application program support) Timestamp (Master functionality)
Redundancy function	supported (Requires host application program support)
Baud rate	Fixed values from 9,6 kBit/s to 12 MBit/s Auto-detection mode is not supported.
Data transport layer	PROFIBUS FDL

Firmware/stack available for netX

netX 10, 50, 51, 52	no
netX 100, netX 500	yes

PCI

DMA Support for PCI targets	yes

Slot Number

Slot number supported for	CIFX 50-DP

Configuration

Configuration by tool SYCON.net (Download or exported configuration file named config.nxd).

Configuration by tool SyCon (exported configuration file named `config.dbm`).

Configuration by packets to transfer bus and slave parameters.

Diagnostic

Firmware supports common and extended diagnostic in the dual-port-memory for loadable firmware.

Limitations

DPV2 isochronous mode and slave-to-slave communication are not supported.

Redundancy functionality not supported for configuration with packets.

1.7 Terms, Abbreviations and Definitions

Term	Description
AP	Application on top of the Stack
AREP	Application Reference End Point
MS0	Master to Slave cyclic communication
MS1	Master (class1) to Slave acyclic communication
MS3	Master (class2) to Slave acyclic communication
DL	Data Link Layer
FSPMM	Fieldbus Service Protocol Machine Master
MSCS1M	Master Class3 to slave clock sync state machine
MSCY1M	Master slave cyclic state machine
DMPMM	Data Link Layer Protocol Machine Master
MSAC1M	Master Class1 to Slave acyclic State Machine
MSAC2M	Master Class2 to Slave acyclic State Machine

Table 2: Terms, Abbreviations and Definitions

All variables, parameters and data used in this manual have basically the LSB/MSB (“Intel”) data representation. This corresponds to the convention of the Microsoft C Compiler.

1.8 References to Documents

This document refers to the following documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX based products, Revision 12, english, 2012.

[2] IEC 61158, edition 2010.

[3] PROFIBUS International: PROFIBUS document #0.032: Normative Parts of PROFIBUS-FMS,-DP, -PA according to the European Standard EN 50170 Volume 2, Edition 1.0, March 1998.

[4] PROFIBUS International: PROFIBUS document #2.082: Technical Guideline PROFIBUS DP Extensions to EN 50170 - Version 2.0, April 1998.

[5] Hilscher Gesellschaft für Systemautomation mbH: Application note, Network scan, english, 2016.

Table 3: References to Documents

2 Getting started

2.1 Overview

The PROFIBUS DP Master exchanges input and output data with PROFIBUS DP Slave devices. The input data and the output data is located in a dual-port memory which allows an application to access them.

Before the PROFIBUS DP Master can perform the data exchange with the slaves, the PROFIBUS DP Master requires a configuration. To configure the PROFIBUS DP Master, you can use

- the configuration software SYCON.net or
- the API of PROFIBUS DP Master, see section *Configuration of Bus and Slave Parameters* on page 54.

The PROFIBUS DP Master provides status information about the PROFIBUS network and the PROFIBUS DP Slaves in the dual-port memory, see section *Status Information* on page 70.

The API of the PROFIBUS DP Master allows the application to use acyclic communication which is read and write data and alarm handling. These topics are described later in this documentation.

2.2 Task Structure

The figure below shows the internal structure of the tasks which together represent the PROFIBUS DP Master Stack.

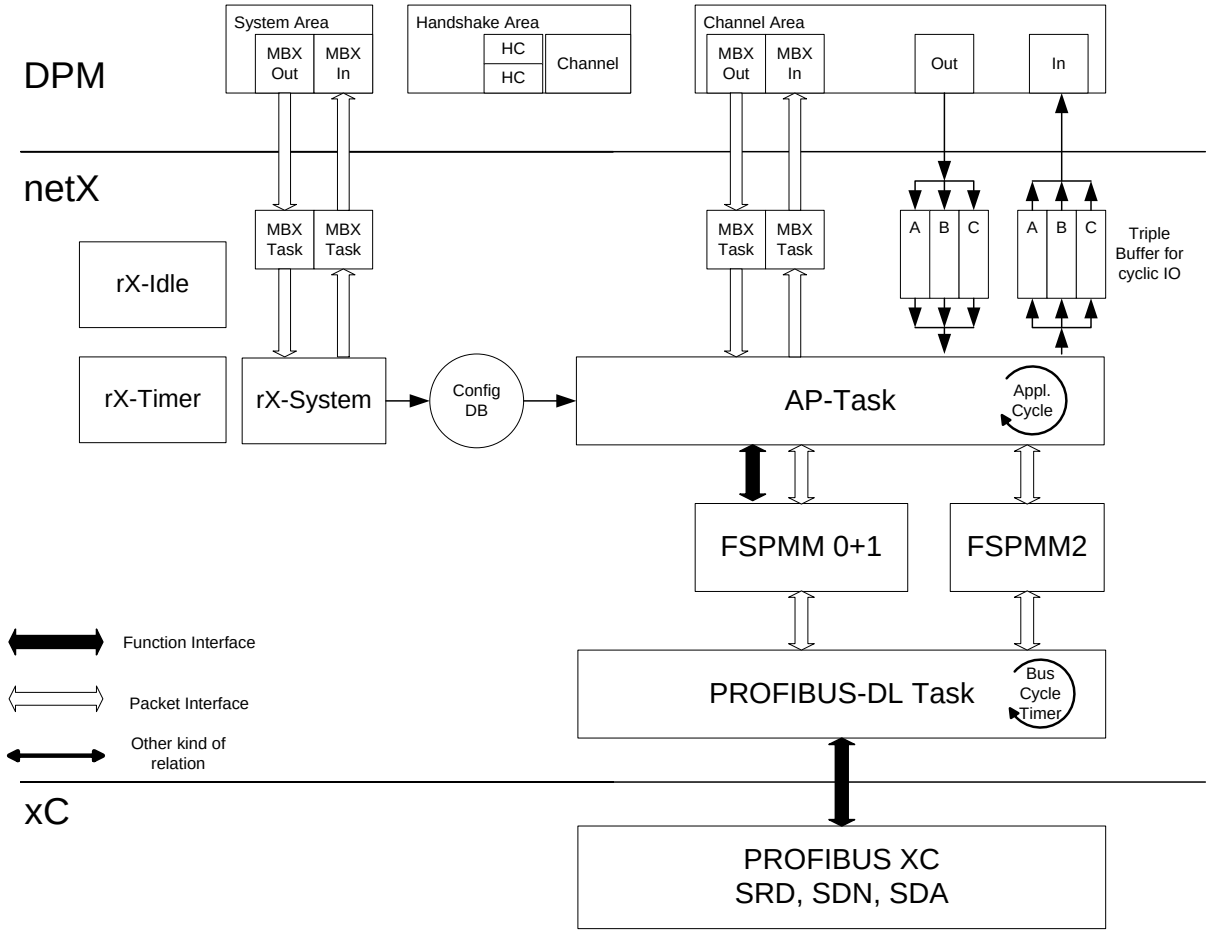

Figure 1: Internal Structure of PROFIBUS DP-Master Firmware

The dual-port memory is used for exchange of information, data and packets. Configuration and IO data will be transferred using this way.

The user application only accesses the task located in the highest layer namely the AP-task which constitute the application interface of the PROFIBUS DP Master Stack.

The PROFIBUS FSPMM task, FSPMM2 task and PROFIBUS DL task together represent the core of the PROFIBUS DP Master Stack.

In detail, the various tasks have the following functionality and responsibilities:

AP task

The AP task provides the interface to the user application and the control of the stack. It also completely handles the Dual Port Memory interface of the communication channel. In detail, it is responsible for the following:

- Handling the communication channels DPM-interface
- Process data exchange
- Channel mailboxes
- Watchdog
- Provides Status and diagnostic
- Handling applications packets (all packets described in Protocol Interface Manual)
- Configuration packets
- Packet Routing
- Handling stacks indication packets
- Provide information about connection state
- Preparation of configuration data
- Evaluation of data base files

PROFIBUS FSPMM task

The PROFIBUS FSPMM task is responsible for the PROFIBUS V0 and PROFIBUS V1 Class1 functionality. In detail, it handles the following items:

- Cyclic Communication
- Acyclic Communication DP V1 Class1
- Alarm handling

PROFIBUS FSPMM2 task

The PROFIBUS FSPMM2 task is responsible for the PROFIBUS V1 Class2 functionality. In Detail, it handles the following items:

- Acyclic Communication DP V1 Class1
- Master-to-Master communication

PROFIBUS DL task

The PROFIBUS DL task is responsible for the FDL services. It provides the following items:

- SAP handling
- Life List
- Data transmission services (SDA, SDN, SRD)

3 PROFIBUS Functions

3.1 PROFIBUS DP and the OSI/ISO Layer Model

As the picture below illustrates, PROFIBUS DP does not affect the layers 3, 4, 5 and 6 of the OSI/ISO reference model of data communication within a network. It only specifies

- Layer 1 (Physical layer)
- Layer 2 (Data link layer)
- Layer 7 (Application layer)

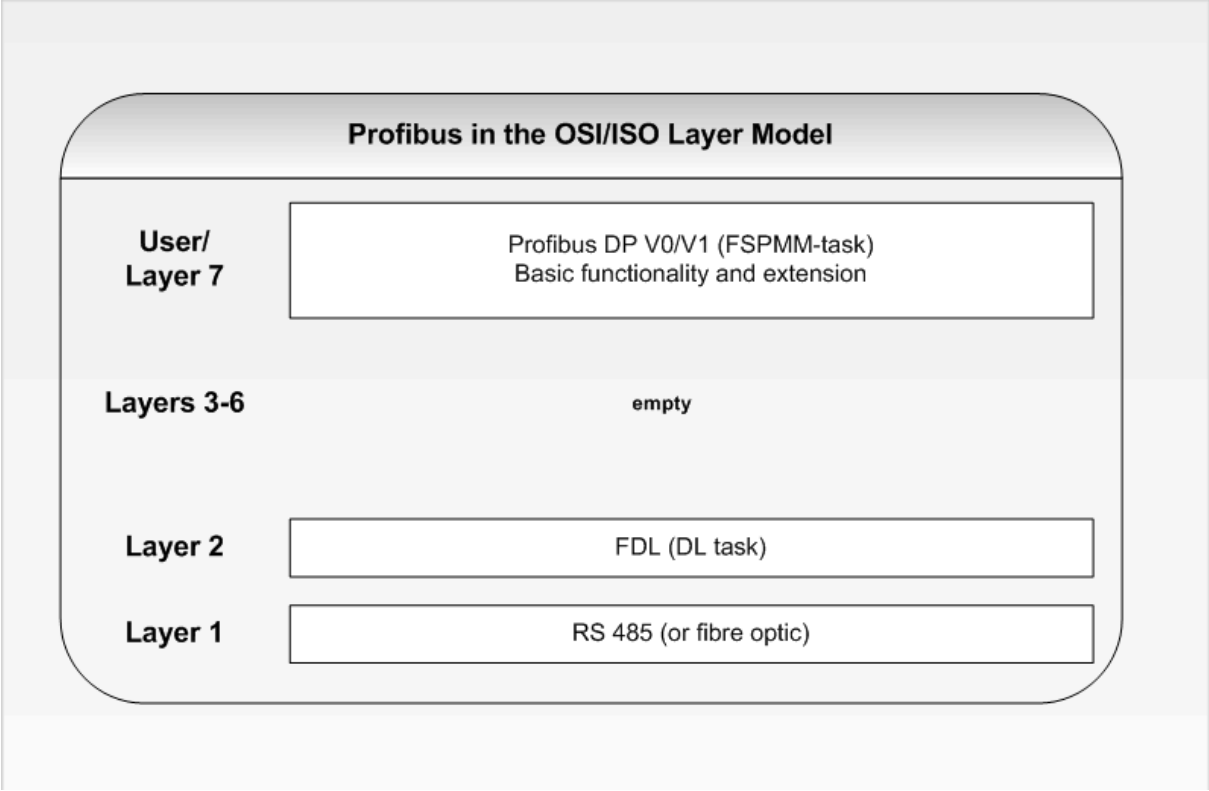

Figure 2: PROFIBUS in the OSI/ISO Layer Model

Above layer 7 the user area begins.

The DL-task is located on layer 2.

The FSPMM-task is located on and above layer 7.

3.2 PROFIBUS DP Operation Modes (States)

A PROFIBUS DP master can be in one of five different states, which are called the operation modes and have a different degree of allowed functionality. These states and their symbolic names are:

Operation Mode	Description
OFFLINE (USIF_OFFLINE)	In OFFLINE state, there is no communication (data transfer) permitted at all. This is the state after initialization. This means, the master is waiting for a signal to start and does not participate in the token ring of the PROFIBUS access control mechanism.
STOP (USIF_STOP)	In STOP state, there is no data transfer permitted between master and slaves. Data transfer to other masters in multi-master system is allowed, however. The bus parameter set has been loaded successfully in order to get into STOP state.
CLEAR (USIF_CLEAR)	In CLEAR state, the master is able to read the input data from the DP slaves. The master forces the outputs to the slaves to be in a safe state (i.e. they contain only the value 0). For instance, incorrect data transfer of a slave can cause the PROFIBUS DP master to fall back from OPERATE state to CLEAR state. Parameterization and configuration checks are possible in this state.
OPERATE (USIF_OPERATE)	In OPERATE state, unrestricted data transfer is possible. This data transfer is cyclic, i.e. periodically, the input values are read from the slaves and the output data are written to the slaves.
PASSIVE (USIF_PASSIVE)	In PASSIVE state, the PROFIBUS DP master only acts as backup master. For more information, see section *Redundancy Functionality* on page 52.

Table 4: PROFIBUS DP Operation Modes (States)

Changes of the operation mode are supervised by an internal state machine within the PROFIBUS DP master.

The system behavior on the failure of a slave can be parameterized in the DP master, so that it switches over automatically from OPERATE condition to the STOP condition. This also interrupts user data transfer to all slaves and the module outputs are switched to the secure zero condition.

A change of the mode is indicated to the AP-task by the indication PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication.

If the AP-task wants to change the mode, it can accomplish this by sending the *PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode* packet, which is described in section 6.1.3.

3.3 Functionality of the FSPMM-Task (Layer 7)

PROFIBUS DP provides the following functionality with its layer 7 part, which is implemented within the FSPMM-task:

- Cyclic Data Transfer
- Acyclic Data Transfer DP V1 Class1
- Configuration of Inputs and Outputs at Slaves
- Alarm Processing
- Diagnostics

3.3.1 Cyclic Data Transfer

Cyclic data transfer is the main task of a field bus system. In PROFIBUS DP, this functionality is located in level DP V0.

3.3.1.1 Description of Processes during Cyclic Data Transfer

The most important prerequisite for cyclic data transfer is, that the operation mode is OPERATE.

In state CLEAR only a restricted cyclic data transfer is possible, in the other states no cyclic data transfer at all is performed.

For example, a data transfer from the master to the slave using the *PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output* packet works in the following way:

- The master sends an SRD request with a variable data field length containing the output data to the slave.
- Immediately the slave will send back an SRD response containing its input data.

The number of available input and output channels is defined at the configuration during system start-up. This configuration is done as a part of the processing of the bus parameter set, for more information see

- Section *Detailed Description of Slave Parameters* on page 62, especially section *Cfg_Data_Len* on page 67 and Cfg_Data on page 67.
- Section *PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register* on page 81.
- Section *PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set* on page 240.

3.3.1.2 Supervision by Watchdog Timer

Cyclic data transfer is supervised by a watchdog timer. The PROFIBUS DP slave's outputs will be switched to a safe state if no regular data transfer happens within in the timer interval of the watchdog timer. The PROFIBUS DP master contains a timer for each DP slave. The reaction of the system depends on the value of the `Auto_Clear` configuration parameter in the following way:

- `Auto_Clear` = TRUE: If one slave supervised by the DP master fails, the outputs of all DP slaves which are supervised by this DP master will be set to the save state. This option offers the highest degree of security.
- `Auto_Clear` = FALSE: If one slave supervised by the DP master fails, the cyclic data transfer is continued and a user-specific reaction can occur. This option enables the addition and removal of stations during full operation of the PROFIBUS system, which might be desirable in many cases.

For more information on the `Auto_Clear` flag, see section *Bp_Flag* on page 59.

Cyclic data transfer can also partially or totally be influenced by the synchronization commands 'sync' and 'freeze'.

This can be done individually for inputs and outputs or also in a combined manner.

3.3.1.3 Synchronization of Inputs

Sending a 'Freeze' command to a DP slave will cause reading the inputs and freezing them. This means, all following read attempts of this input will deliver the value read at the time of the 'Freeze' command as long as the 'Freeze' command has not been suspended. Suspending the 'Freeze' command is possible either by an 'unfreeze' command or by a subsequent 'Freeze' command.

Sending an 'Unfreeze' command to a DP slave will cause normal operation of the cyclic inputs again.

3.3.1.4 Synchronization of Outputs

Sending a 'Sync' command to a DP slave will cause the current output values to be frozen. These values will be used until the ''Sync' command is suspended. This can be accomplished either by sending an 'Unfreeze' command or another 'Sync' command.

Sending an 'Unsync' command to a DP slave will cause normal operation of the cyclic outputs again.

Also all outputs can be set to the safe state by the Clear_Data option. All these options are provided by the global control command; see section *PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF – Global Control Message* on page 125.

3.3.2 Acyclic Data Transfer

Acyclic data transfer only happens on request and not periodically in cycles as the cyclic data transfer.

In PROFIUS DP, acyclic data transfer is supported by level DP V1, but not by the lowest level DP V0. This means it is necessary that the slave to read from or write to supports the DP V1 extensions to the PROFIBUS standard. The acyclic data transfer happens with lower priority than the cyclic data transfer does as it uses gaps, i.e. available remaining times of the bus cycles which have not been required for cyclic data exchange.

3.3.2.1 Acyclic Data Transfer of the DP Master Class1

An error on DP V1 Class1 acyclic communication could affect the cyclic data exchange communication – DP V0 i.e. restart of cyclic communication with particular slave.

To use the described packets the following requirements has to be fulfilled:

- Master must be in cyclic data exchange with the slave
- The slave must support the DP V1 Class1 and the feature must be activated in configuration

Only one active request per slave is supported. To send a new one, the confirmation of the previously request is necessary to return.

The main packets for DP V1 Class1 acyclic data transfer are:

Packet	Page
PROFIBUS_FSPMM_CMD_READ_REQ/CNF – DP V1 Class1 Read	101
PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – DP V1 Class1 Write Request	105

Table 5: Packets for DP V1 Class1 Acyclic Data Transfer

With these packets, data areas within single modules of the slave can be accessed. These data areas can be addressed by a slot and index based mechanism:

The `ulSlot` parameter is used (both for read and write access) to determine the slot in the destination device. Slot in this context usually simply means a single module of the destination device. The allowed range for this parameter extends from 0 to 254 as the value 255 is reserved by the PROFIBUS DP V1 specification.

The `ulIndex` parameter is used (both for read and write access) to determine the desired data block in the desired module of the destination device. The allowed range for this parameter extends from 0 to 254 as the value 255 is reserved by the PROFIBUS DP V1 specification.

The length of the data area to be read can be specified by the `ulLength` parameter but it may not exceed the upper limit of 240 bytes. The applied error handling is described below in section *DP V1 Error Processing* on page 18.

The values for slot, index and the corresponding data sets must be referred by the slave manual and/or GSD file, as they are manufacturer specific.

3.3.2.2 DP V1 Error Processing

The error handling is similar for read and write access. It is described in section 10.3.1 “Meaning of Error_Code_1 and Error_Code_2 at DPV1 mode” of the document *“Technical Guideline, PROFIBUS DP Extensions to EN 50170 - Version 2.0*” published by PNO under order 2.082. This applies both to DP V1 Class1 and DP V1 Class2. In case of error, two Error_Codes are delivered in the data area. These two Error_Codes represent further detailed error information.

Error_Code_1

D7	D6	D5	D4	D3	D2	D1	D0
Error_Class		Meaning		Error_Code			
0 to 9 =		reserved					
10 =		application		0 = read error 1= write error 2 = module failure 3 to 7 = reserved 8 = version conflict 9 = feature nor supported 10 to 15 = user specific			
11 =		access		0 = invalid index 1 = write length error 2 = invalid slot 3 = type conflict 4 = invalid area 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10 to 15 = user specific			
12 =		resource		0 = read constraint conflict 1 = write constraint conflict 2 = resource busy 3 = resource unavailable 4 to 7 = reserved 8 to 15 = user specific			
13 to 15 =		user specific					

Table 6: Explanation of Error Class and Error Code within Error_Code_1

Error_Code_2 is fully user specific and thus cannot be explained in more details.

3.3.3 Configuration of Inputs and Outputs at Slaves

This is a process happening at the slaves, but requiring some supervision and control by the master. It is therefore useful to have a look to the slave:

At the PROFIBUS DP Slave, there are three important situations (and packets associated with these) dealing with the configuration blocks:

- At the initialization of the state machine for cyclic data processing (MSCY1S).
- When the AP task requests a change in the 'Is'-configuration data using a 'Set Cfg' command.
- When the slave receives a `PROFIBUS_FSPMS_CMD_CHECK_CFG_IND` indication from the master.

The first packet initializes the state machine for cyclic data processing at the slave and provides the current configuration data to use from the initialization of cyclic data transfer until the first request to change the parameter set if one occurs. The configuration data (so called "Is-configuration data" or "Real-configuration data") is stored in the slave and heavily influences the operation of the slave. It is transferred to the slave by the parameter `abRealCfgData[…]` of the slave's cyclic initialization packet and uses the data format described precisely below.

The second packet allows changing the "Is-configuration data" stored at the slave after initialization if necessary. In this case the "Is-Configuration data" to be set is provided in the parameter `abCfgData[…]`. The format is exactly the same as during initialization, see below.

The third packet indicates a request from the PROFIBUS DP Master to compare the real configuration data (i.e. those stored in the slave) with the configuration data which the master assumes to be correct. These are called the "assumed configuration data" and transmitted from the master with a check configuration indication in the parameter `abCfgData[…]`.

All these packets share the same formats describing the configuration of the modules' inputs and outputs.

3.3.3.1 Format of PROFIBUS DP Configuration Data

The rest of this section describes the structure of the parameter block containing the configuration data for the slave(s), which decides on the number of input and outputs of the slave. This data block is sent from the PROFIBUS DP master to the PROFIBUS DP slave with the command 'Check_Cfg' to force the slave to compare this configuration with its own internally saved one.

In detail, it is possible to specify here:

- Input data and their size
- Output data and their size
- Manufacturer-specific data and their size
- The amount of consistency to apply

Note: Consistency in this context means whether the whole data need to be interpreted as an entity or each byte/word may separately be interpreted by the PROFIBUS DP Master.

The parameter block consists of

- An identifier byte (either in the general or in the special identifier format, see below)
- A length byte

The length of the complete check configuration data block must not exceed 244 bytes, however, it is recommended not to exceed an amount of 32 bytes otherwise some restrictions apply, see the underlying IEC 61158 or EN 50170 specification.

3.3.3.2 Identifier Byte for the General Format

The identifier byte can be specified in the general (also called compact) format or the special format. In the general format, the meaning of the single bits is defined as follows:

General Identifier Format of Identifier Byte (according to IEC 61158/EN 50170 Specification)				
Bit	**Name**	**Value**	**Description**	
D7	Consistency	Consistency extends over		
		0	Byte or word (in case of D6 = 0)	
		1	Whole length (in case of D6 = 1)	
D6	Length format	Length format		
		0	Byte structure	
		1	Word structure	
D5	Output	Input/Output/Special identifier format		
		0	D4=0	This combination signifies the special identifier format, see below.
			D4=1	Input
		1	D4=0	Output
			D4=1	Input- Output
D4	Input		See above (D5). D5 and D4 are evaluated together.	
D3	Length of data	0	1 Byte/Word	
		1		
		…		
		15	16 Bytes/Words	
		(choice of byte or word depends on length format bit)		

Table 7: General Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification

Note: When transferring data in word mode, the high byte is transferred first by PROFIBUS DP, then the low-byte. However, the PROFIBUS DP Master has the possibility to swap this sequence of the bytes within the word, if required by the target system.

3.3.3.3 Identifier Byte for the Special Format

To allow extended configurations and to increase flexibility, a special extension of the identifier system described above is also supported by PROFIBUS DP. The main advantages of this format are:

- It is possible to determine the number of input and output bytes associated to the defined identifier.
- User specific data can be added.

This format is called the special identifier format and signified by the combination of byte 4 and 5 both being zero as already described above in the discussion of the general identifier format.

Special Identifier Format of Identifier Byte (according to IEC 61158/EN 50170 Specification)				
Bit	**Name**	**Value**	**Description**	
D7.. D6	Consistency/ Length format (used for Input/Output)	Input/Output		
		D7	D6	
		0	0	Free area
		0	1	1 length byte for inputs follows
		1	0	1 length byte for outputs follows
		1	1	1 length byte for outputs and 1 length byte for inputs follows
D5.. D4	Signification	Signification of Special identifier format		
		D5	D4	(No other combinations allowed in Special Identifier Format)
		0	0	This combination signifies the special identifier format, see below.
D3	Data Length	Length of manufacturer specific data		
		0	No manufacturer specific data follow; no data in Real_Cfg_Data.	
		1-14	Manufacturer specific data of the length specified in the following byte(s) follow, these should be equal to those in Real_Cfg_Data.	
		15	In case of Check_Cfg: No manufacturer specific data follow, verification may be omitted.	

Table 8: Special Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification

3.3.3.4 Length Byte

The length bytes following the special identifier format bytes are organized as described in the table below:

Structure of Length Byte in the Special Identifier Format of the Identifier Byte according to IEC 61158/EN 50170 Specification			
Bit	**Name**	**Value**	**Description**
D7	Consistency	Consistency extends over	
		0	Byte or word (in case of D6 = 0)
		1	Whole length (in case of D6 = 1)
D6	Length format	Length format	
		0	Byte structure
		0	Word structure
D5..D0	Length of data	0	1 Byte/Word
		1	2 Bytes/Words
		…	
		63	64 Bytes/Words

Table 9: Structure of Length Byte in the Special Identifier Format of the Identifier Byte according to IEC 61158/ EN 50170 Specification

3.3.4 Alarm Processing

3.3.4.1 PROFIBUS DP V1 Alarm Concept

Alarms are messages from the slave to the server which are for instance caused by extraordinary events within a specific module (slot) of the slave and which require processing with high priority and an explicit acknowledgement. The reception of the alarm message from the slave causes a PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND indication at the master.

The aforementioned explicit acknowledgement must be sent from the master to the slave (PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ) which subsequently will confirm the reception of the acknowledgement again by either a positive or negative confirmation message (PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF or PROFIBUS_FSPMM_CMD_ALARM_ACK_NEG_CNF).

The importance of this explicit acknowledgement results from the intention to securely avoid overwriting of alarms: As no unacknowledged alarm may be cleared, overwriting of pending alarms is prohibited. In addition, the alarm indication is stored within an internal queue at the DP Master. From this queue, it may be removed not earlier as the alarm has been acknowledged.

Technologically alarms work similarly as diagnostic messages where status information and alarm-specific information is stored in the area which otherwise has been reserved for manufacturer-specific data.

Alarms in PROFIBUS DP V1 may only be processed by PROFIBUS DP Class1 Masters.

3.3.4.2 Alarm Types

The following kinds of alarms are available:

- **Diagnostic_Alarm**: Indicates a special event such like short circuit, over temperature.
- **Process_Alarm**: Indicates a special event in the supervised process e.g. reaching a critical limit of a supervised value.
- **Pull_Alarm**: Indicates that a specific module has been pulled out.
- **Plug_Alarm**: Indicates that a specific module has been plugged in.
- **Status_Alarm**: Indicates a change of state of a specific module, such as ‘run’, ‘stop’ or ‘ready’, for instance.
- **Update_Alarm**: Indicates the value of a parameter in a module has been changed by a local operation or remote access.
- Furthermore, manufacturer-specific alarms may be defined.

These are denominated as the alarm types (represented by the `ulAlarm_Type` variable of all alarm related packets)

3.3.4.3 Conditions for Alarm Indication

The following conditions need to be fulfilled in order to cause an alarm indication at the DP master:

- The slave must be ready for (cyclic) data exchange (DATA-EXCH mode).
- An acyclic connection (MSAC_C1 connection) must have been activated (`DPV1_Enable` = TRUE).
- The corresponding alarm type has been enabled
- The maximum number of pending alarms has not been exceeded. This maximum number can be dined either as a limit of totally allowed alarms independently from the alarm type or there is only one alarm of each specified type permitted.

3.3.4.4 Contents and Structure of the Alarm Message

According to the PROFIBUS DP standard, the content of the alarm message is a diagnostic message. It can consist by one, multiple or all of the following items:

- Alarm PDU
- Status-PDU (not discussed here)
- Identification-related diagnosis
- Channel-related diagnosis
- Revision number

At least one item is required to be chosen.

The Alarm PDU is set up as a block of up to 64 bytes including a 4-byte header:

The first byte of this block is header byte. Its first two most significant bytes are set to zero indicating this is an alarm message. The rest of contains the length of the alarm message in bytes (up to 63 bytes are available here).

The second byte is the alarm type as discussed above. It is coded as described in section *PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification* on page 110, see *Table 66: Alarm Types.*

The third byte contains the number of the affected module.

The fourth byte is the alarm specifier. It is coded as described for the variable `ulSpecifier` of packet PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND.

Bytes 5 to 63 of the alarm message provide an area for transparently delivering the received diagnostic data of the slave, As these data are slave dependent please refer to the documentation of the slave for more information. This area is represented by variable `abDiagnostic_User_Data[]` in packet PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND.

Identification-related diagnosis and channel-related diagnosis are described in more detail in section *Diagnosis* on page 26.

For more information, please refer to

- Section PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification on page 110.
- Section PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge on page 113.

3.3.5 Diagnosis

The PROFIBUS DP Master firmware offers two functions for making diagnostic information from the slaves available at the server:

- PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic
- PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic

If a new diagnostic is available, the PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic indication will inform the master about this fact, and the master can decide whether and when to request a diagnostic block from the slave from which the indication originated. This is done subsequently with the PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic packet which will be confirmed.

This section discusses the meaning of the diagnostic information delivered by the PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic packets.

However, the diagnostic features of PROFIBUS DP depend on the version (V0 or V1). While DP V0 offers a standard diagnosis, DP V1 provides extended diagnostic capabilities.

3.3.5.1 Diagnostic model of PROFIBUS DP V0

The diagnostic model of PROFIBUS DP V0 provides diagnostic messages consisting of 6 bytes (denominated as octets in the PROFIBUS standard) of information on the cyclic data traffic.

The meaning of these octets is:

Octet 1: Station Status_1

Bit	Description	Meaning
D0	Station_Non_Existent	indicates that the slave does not exist or is not responding. An operating slave always sets this bit to 0.
D1	Station_Not_Ready	indicates that the slave is not ready for cyclic data transfer.
D2	Cfg_Fault	indicates a configuration fault: the slave has been parameterized wrongly. The slave's internal configuration data differ from those the master has sent to the slave.
D3	Ext_Diag	indicates the area `'Ext_Diag'` is used for extended diagnostic according to the PROFIBUS DP V1 extensions. More than the 6 octets of standard diagnostic data contain valid and relevant diagnostic data. For interpretation, see section 3.3.5.2 *Diagnostic model of PROFIBUS DP V1*.
D4	Not_Supported	indicates that an unknown command has been detected by the slave. The requested command is not supported by the slave.
D5	Invalid_Slave_Response	indicates that the response of the slave was invalid or not plausible. An operating slave always sets this bit to 0.
D6	Prm_Fault	indicates that the last received parameter telegram was defective or incorrect.
D7	Master_Lock	indicates that the slave has been parameterized by another master, so this master is not permitted to control the requested slave. An operating slave always sets this bit to 0.

Table 10: Octet 1: Station Status_1

Octet 2: Station Status_2

Bit	Description	Meaning
D0	Prm_Req	indicates that the slave requires new parameterization and configuration As long as a new parameterization has not been performed this bit remains set to the value ‘1’.
D1	Stat_Diag	indicates that the master needs to fetch diagnostic information from slave, until this bit is released. If the slave is not able to deliver valid output data, it will set this bit.
D2	Fixed 1	indicates DP V1 operation.
D3	WD_On	indicates that the watchdog timer supervision mechanism for the slaves is activated.
D4	Freeze_Mode	indicates that the slave has received a *‘Freeze’* command and since then no *‘Unfreeze’* command.
D5	Sync_Mode	indicates that the slave has received a *‘Sync’* command and since then no *‘Unsync’* command.
D6	Reserved	reserved by the DP standard.
D7	Deactivated	The slave has not been projected. A projected slave always sets this bit to 0.

Table 11: Octet 2: Station Status_2

Octet 3: Station Status_3

Bit	Description	Meaning
D0- D6	Reserved	reserved.
D7	Ext-Diag_Overflow	indicates that the slave has more diagnostic data available than it can send.

Table 12: Octet 3: Station Status_3

Octet 4: Master_Add

This byte contains the address of the master, which has done the parameterization of the slave. If a slave has not been parameterized, this value is set to 255.

Octet 5_6: Ident_Number

In these bytes, the slave station reports its identification number, which has been assigned to it by the manufacturer.

3.3.5.2 Diagnostic model of PROFIBUS DP V1

The diagnostic model of PROFIBUS DP V1 is an extension to that of DP V0. It provides three kinds of messages informing the master about the situation of the slaves:

- Diagnostic messages
- Status messages
- Alarm messages

This section discusses diagnostic messages. Alarm messages are discussed in a separate section due to their importance and their separate handling. Status messages are not relevant in the context of this manual.

While diagnostic messages and status messages are buffered in the PROFIBUS DP system, alarms are queued in order to prevent them from being overwritten.

If the **Ext_Diag** bit has been set within octet 1, there is more than 6 bytes of diagnostic data available according to the DP V1 standard

The diagnostic data area may contain (additionally to the mandatory 6 octets):

- Alarm PDU (see section 3.3.4.4 “Contents and Structure of the Alarm Message”)
- Status-PDU (not discussed here)
- Device-related diagnosis
- Identification-related diagnosis
- Channel-related diagnosis
- Revision number

For diagnostic blocks, the following general rules apply:

1. Each of these diagnostic blocks consists of a 1-byte header determining type and length of the diagnostic block and the data part of the block.
2. The length of one block is limited to 64 bytes including the header,
3. The length of all blocks together is limited to 238 bytes.

The first byte of each block is the header byte. Its first two most significant bytes code the type of diagnostic block. The meaning of bits D0 to D5 depends on the type of diagnostic block according to the following *Table 13: Diagnosis Header*.

The structure of the diagnosis header is :

D7	D6	D5	D4	D3	D2	D1	D0	Description
0	0	0-63: Length of diagnostic block (in bytes)						Indicating device-related diagnosis
0	1	0-63: Identifier_Number of related module						Indicating identification-related diagnosis
1	0	0-63: Number of affected module (1-64)						Indicating channel -related diagnosis

Table 13: Diagnosis Header

Device-related diagnosis

In DP V0, the interpretation of device-related diagnostic information depends on the contents of the `*.GSD` file of the device.

In DP V1, device-related diagnostic blocks will be either alarm or status blocks, so they do not need to be discussed here.

Identification-related diagnosis

For identification-related diagnosis, the structure of the header is explained in *Table 13: Diagnosis Header.*The following bytes indicate which module of the slave has a diagnosis. For 8 modules, one byte is used to indicate this:

D7	D6	D5	D4	D3	D2	D1	D0	Description
							x	Module 1 has a diagnosis available.
						x		Module 2 has a diagnosis available.
…								
x								Module 8 has a diagnosis available.

Table 14: Identification-related diagnosis – First Data Byte

If 16 modules have been used, the next byte would be then

D7	D6	D5	D4	D3	D2	D1	D0	Description
							x	Module 9 has a diagnosis available.
						x		Module 10 has a diagnosis available.
…								
x								Module 16 has a diagnosis available.

Table 15: Identification-related diagnosis – Second Data Byte

Note: The module numbers applied here relate to the order of the modules within the configuration data telegram.

Channel-related diagnosis

The channel-related diagnosis is more detailed compared to the identification-related diagnosis and reports about channel errors within the modules of a slave.

For each channel an own diagnostic block consisting of a 1 byte header according to *Table 13: Diagnosis Header* and 2 data bytes is used.

The second byte contains the following information

D7	D6	D5	D4	D3	D2	D1	D0	Description
0	1	0-63: Number of channel						Channel is an input-channel
1	0	0-63: Number of channel						Channel is an output-channel
1	1	0-63: Number of channel						Channel is an input- and output-channel

Table 16: Channel-related Diagnosis – First Data Byte

The third byte contains the cause of a diagnosis event in the module.

D7	D6	D5	D4	D3	D2	D1	D0	Description
0	0	1	0:31: see below					Bit
0	1	0	0:31: see below					2 Bits
0	1	1	0:31: see below					4 Bits
1	0	1	0:31: see below					Byte
1	1	0	0:31: see below					Word
1	1	1	0:31: see below					2 Words
any			1					Short circuit
			2					Lower voltage limit exceeded
			3					Upper voltage limit exceeded
			4					Upper power limit exceeded
			5					Upper temperature limit exceeded
			6					Connection broken
			7					Lower limit exceeded
			8					Upper limit exceeded
			9					Error
			10-15					Reserved
			16-31					Manufacturer-specific

Table 17: Channel-related Diagnosis – Second Data Byte

3.4 Functionality of the FSPMM2-Task

This task provides the functionality of DP V1 Class2 on layer 7 of the OSI/ISO layer model. The acyclic data processing capabilities for PROFIBUS DP Masters Class2 provided by the FSPMM2-task are described in this section.

The error handling is the same as already described for DP V1 Class1, so please refer to section *DP V1 Error Processing* on page 18.

3.4.1 Overview of Supported Functionality

The packets listed below are available for providing these acyclic services of DP V1 Masters Class2. They are described in section *The FSPMM2-Task* on page 145.

Supported Functionality of PROFIBUS DP V1 Class2 Acyclic Communication

Section	Packet name/Title of section
6.2.1	PROFIBUS_FSPMM2_CMD_INIT_REQ/CNF – Initialization Command
6.2.2	PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate
6.2.3	PROFIBUS_FSPMM2_CMD_READ_REQ/CNF – DP V1 Class2 Read Request
6.2.4	PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class2 Write Request
6.2.5	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request
6.2.6	PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection
6.2.14	PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication
6.2.15	PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication

Table 18: Overview of available Packets supporting DP V1 Class2

3.4.2 General Remarks on DP V1 Class2

The DP V1 Class2 technology is characterized by the following features:

- Connection-oriented data transmission over a DP V1Class2 connection
- Connection must be established explicitly (‚initiated') before first data transmission can take place. A communication reference identifies the established connection from that time on.
- After that the services ‚Read', ‚Write' and ‚Data' Transport are available.
- Connection must explicitly be de-established (‘aborted').
- The stack allows only one connection from a master Class2 to a slave at one time
- A slave may have multiple connections to a master Class2 and one to a master Class1 at a time.

3.4.3 DP V1 Class2 Masters

Class2 masters have special capabilities when communicating with slaves such as:

- Reading the configuration data of the DP slave
- Reading the input and output data (but no write access possible)
- Address assignment to slaves

Class2 masters are used for commissioning the network and for maintenance and diagnostic purposes and may be removed after the start-up phase when the system is working correctly.

Class2 masters acyclically communicate with slaves over a Class2 connection (also called MSAC_C2 connection relationship). This Class2 connection must explicitly be set up.

3.4.3.1 Acyclic Data Transfer of the DP Master Class2

DP V1 Class2 communication is independent of a DP V0 communication, but relies on a separate communication channel which must be established by the application.

To use the described packets the following requirements has to be fulfilled:

- Master must be configured and in Token exchange – Bus On
- The slave must support the DP V1 Class2
- Application must be registered to receive the indications

3.4.4 Basic Services for Connection Maintenance

Two services are defined for handling a Class2 connection between a Class2 master and a slave:

- MSAC2_Initiate service.

  This service is used by the DP master Class2 in order to set up a DP V1-Class2 connection to the DP slave. The DP slave then receives a “Initiate” indication. See section *Initialization Process of a DP V1-Class2 Connection (MSAC2_Initiate)* on page 35.

- MSAC2_Abort service.

  This service is used by the DP master Class2 in order to abort a DP V1-Class2 connection to the DP slave. The DP slave then receives an abort indication. The DP slave may also request an abort of the Class2 connection by itself. For instance, this can be accomplished by sending the abort packet.

  (corresponding packets: PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection and PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication)

3.4.5 Basic Services available after Establishing a DP V1-Class2 Connection

The following services are available after a DP V1-Class2 connection has been set up successfully:

- MSAC2_Read

  This service is suited for reading a data block from the specified index of the module specified by the choice of the slot of the specified API.

  (corresponding packet: PROFIBUS_FSPMM2_CMD_READ_REQ/CNF – DP V1 Class2 Read Request)

- MSAC2_Write

  This service is suited for writing a data block to the specified index of the module specified by the choice of the slot of the specified API.

  (corresponding packet: PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class2 Write Request)

- MSAC2_Transport

  This service is suited for reading and simultaneously writing a data block from the specified index of the module specified by the choice of the slot of the specified API.

  (corresponding packet PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request)

Only one of these acyclic services can be executed at a time per MSAC2M connection.

3.4.6 Extended Addressing Mechanism

Principally, DP V1-Class2 uses the same addressing mechanism as DP V1-Class1 described above: All data blocks (length 240 bytes at maximum) permitted for read or write access correspond to a specific module. This means, they can be addressed by

- The slot number (identifying the module)
- The index (identifying the functional block within the module)

The range of module numbers must begin with 1 and it must be contiguous in ascending order. The device itself has slot number 0. But there is an important extension: As an additional layer a DP master Class2 may contain different instances of applications. Each such application process instance (APIs) is associated with a security level (SCL, for instance read-only access) and consists of modules (called slots here) and indexes for which the same rules apply as for the DP master Class1. Which API is used depends on the choice made during initialization of the MSAC2M connection, so this choice determines the API for the whole lifetime of the connection.

3.4.7 Short Description of the MSAC2M State Machine

Within the MSAC2M state machine 3 main states have been defined:

- ‘Power-On’ state
- ‘Closed’ state
- ‘Open’ state

There are also some additional intermediate states defined in which the system waits for some events.

- In ‚Power-On’ state the system waits for initialization by the state machine manager.
- In ‚Closed’ state the system waits for initialization by the ‘Initiate’ request `PROFIBUS_FSPMM2_CMD_INITIATE_REQ`. The system is then changing into an intermediate state and waits there for the confirmation. As soon as positive confirmation takes place, the system changes to the state ‚Opened’.
- Only in ‚Opened’ state the main data transfer services ‚Read’, ‚Write’ and ‚Data Transport’ of DP V1 Class2 are available.

However, at one time only one connection may be held by a DP V1 Class2 master, where as one slave may be connected to several DP V1 Class2 masters and a Class1 master simultaneously. This is also supervised by the state machine. The ‚Opened’ state is left if an abort occurs. 3 different situations can lead to an abort. These are:

- Abort by the user of the master (In this case, the master sends a `PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF` – Request Abort of Connection request to the slave)
- Abort by the user of the slave (In this case, the slave sends an abort request and subsequently the master receives an `PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication` from the slave)
- An automatic abort due to communication problems (protocol error) occurs.

The MSAC2M state machine also deals with some administrative information such as the service access points to be used for low-level operations or some internal timers (see S-Timer and R-Timer below). A complete description of the MSAC2M state machine with all intermediate states, transitions and details is given in section 12.7 of the specification document.

3.4.8 Initialization Process of a DP V1-Class2 Connection (MSAC2_Initiate)

The establishment of an MSAC_C2 connection is done by executing the MSAC2_Initiate service. This is a process with several stages:

- First, the PROFIBUS DP Master Class2 sends a special request (Initiate-REQ-PDU) to a special service access point of the PROFIBUS DP Slave.
- On reception of this request, the slave determines whether there is a free service access point available, and if there is one, which one will be used. This service access point is then provided to the master Class2.
- At the same time the ‘Initiate’ indication is sent from the PROFIBUS DP Slave to the slave application.
- The slave now sends a message (RM-REQ-PDU) to the master.
- This message is received at the master. The service access point will be stored then.
- The master now acknowledges that it wants to establish a connection using the service access point chosen by the slave. In this context you can imagine a service access point as a kind of communication channel between master and slave.
- The master now waits for the positive or negative reaction of the slave in a polling mode.
- The slave will react on reception of the response to the ‘Initiate’ indication.
- If a positive reaction is received from the slave, the master finally opens the connection over the chosen service access point and the slave reacts.

This process is illustrated by the picture below.

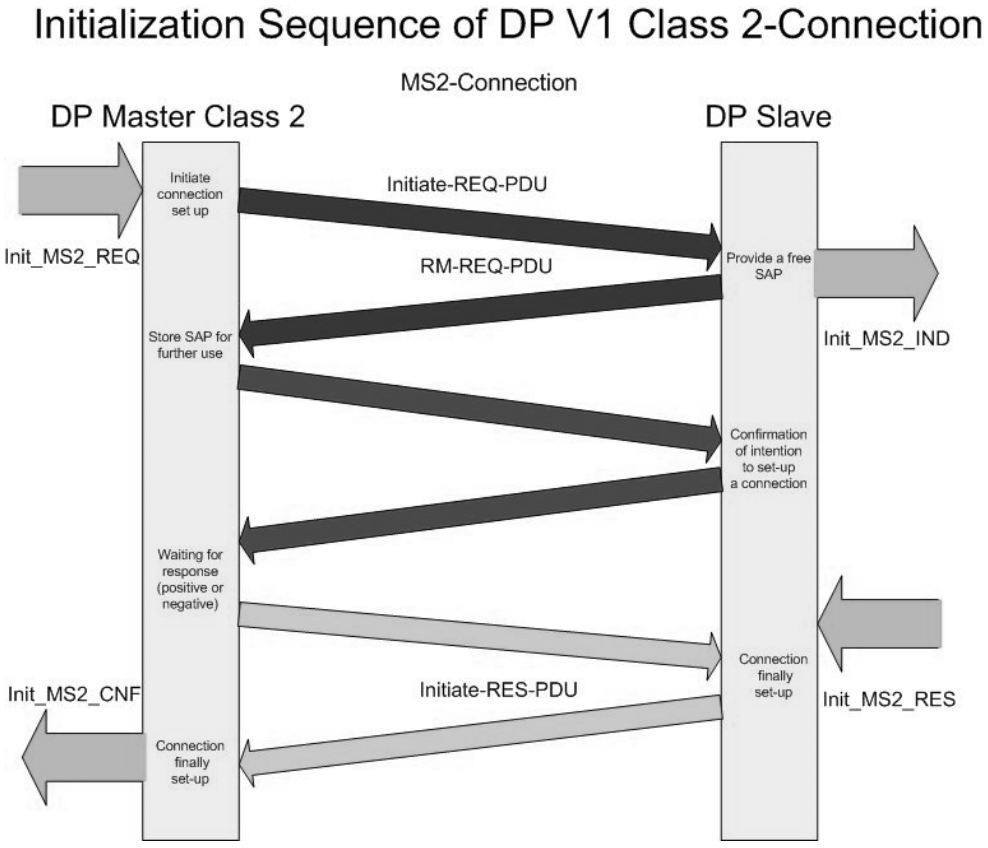

Figure 3: Initialization Sequence of DP V1 Class2-Connection

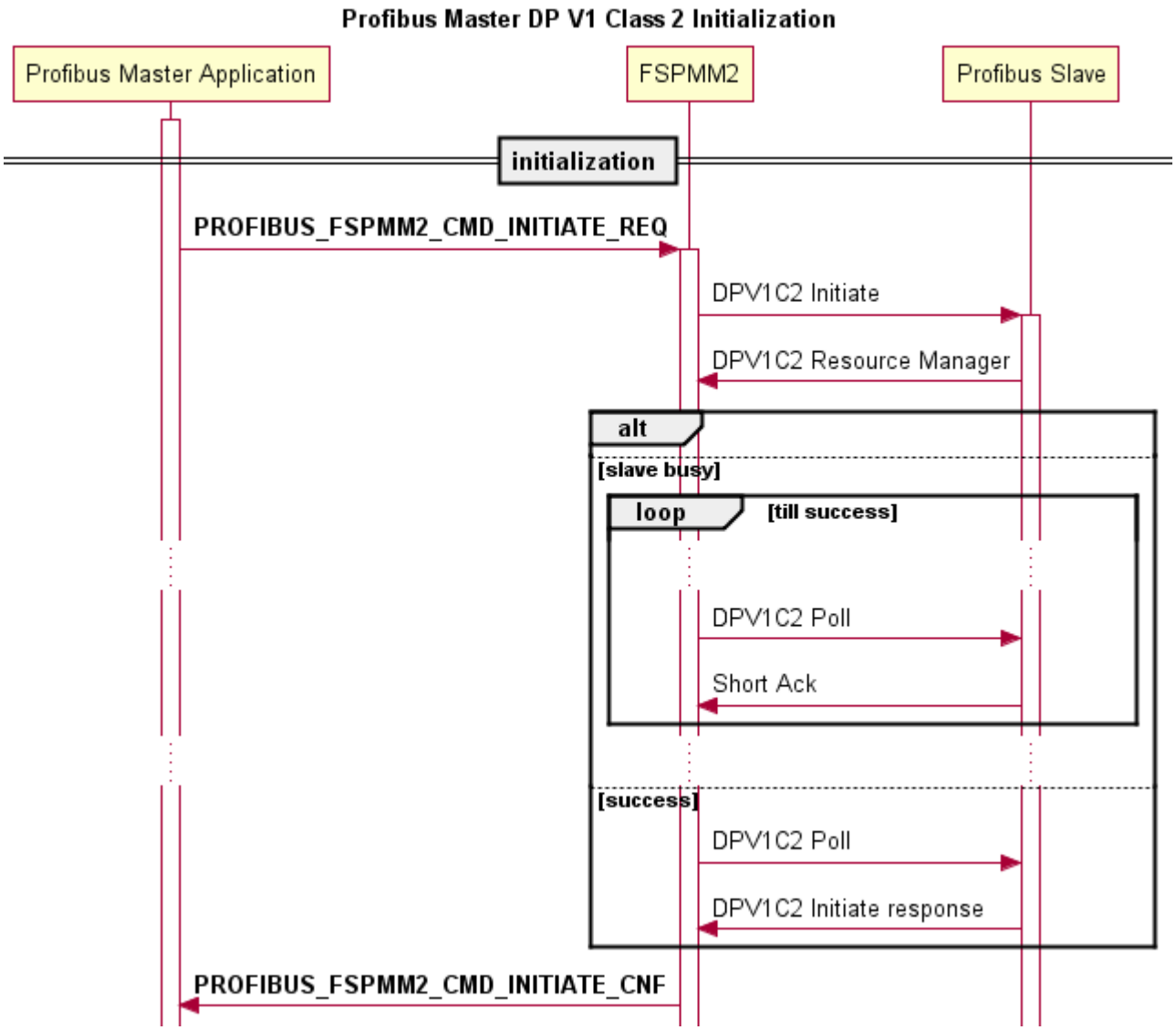

Figure 4: Profibus Master DP V1 Class2 Initialization

3.4.9 Detailed Description of DP V1 Class2 Initialization Parameters

The following parameters are relevant when establishing an MSAC_C2 connection.

Parameter	Meaning	Range of Values	Default Values
ulRemAdd	Slave address of slave to be configured	0…125	
usSendTimeout	Time to monitor Class2 connection (10ms time base)	10-65535	200 (2000 ms)
bFeaturesSupported1	Supported service functionality: Bit 0: Read_Write, Bit 1-7: reserved	0x00-0x01	0x01
bFeaturesSupported2	Reserved	0x00	0x00
bProfileFeaturesSupported1	Supported profile specific service functionality. Please refer profile specific documentation	0-255	0x00
bProfileFeaturesSupported2	Supported profile specific service functionality. Please refer profile specific documentation	0-255	0x00
usProfileIdentNumber	Profile specific unique identifier. All identifiers are defined from a PNO pool.	0-65535	0 (no profile)
tAddAddrParam	Additional address information		

Table 19:DP V1 Class2 Initialization Parameter

Features supported

DP master Class2 and DP slave communicate about the supported functionality of each other. This gives the slave the opportunity to adjust its functionality to the master's requirement or to reject the request if it cannot fulfill them.

Profile Features supported

DP master Class2 and DP slave communicate about the supported profile features. The meaning of the bits of these two bytes depends on the profile or vendor.

Profile Ident No.

This number allows the unique identification of a profile. This number is assigned by the PNO All device using the same profile definition have to use the same Profile Ident No. The value 0 indicates that no profile is supported. The profile ident number is a 16-bit number.

Additional address information

The additional address information consists of address data both of the source and of the destination. The format of the address data depends on the address type. The following elements are defined:

Parameter	Meaning	Range of Values	Default Values
S_Type	Type of the source address parameter	0-1	0
S_Len	Length of the source address parameter	2-255	2
D_Type	Type of the destination address parameter	0-1	0
D_Len	Length of the destination address parameter	2-255	2
S_Addr	See definition of address. 1st Part of abAddParam		
D_Addr	See definition of address. 2nd Part of abAddParam		

Table 20:DP V1 Class2 Initialization additional address parameter

The additional address parameter consists of two parts providing information about the source and the destination.

For both, the following information is stored within the additional address parameter:

- Additional address information
- Presence (Type = 1) or absence (Type = 0) of network/MAC address
- Length of additional address information

Parameter	Meaning	Range of Values	Default Values
If address type = 0			
API	Application interface (uint8)	0-255	0
SCL	Access level (uint8)	0-255	0
If address type = 1			
API	Application interface (uint8)	0-255	0
SCL	Access level (uint8)	0-255	0
Network Address	Network address (octet string[6])		
MAC Address	MAC Address (octet string[])		

Table 21:DP V1 Class2 Initialization format of address parameter

The additional address information contains the following information:

- The application process instance (API) of the source/destination.
- Access level of the source/destination.
- Network address (optional)
- MAC address (optional)

For these values the following rules apply:

- The application process instance (API) of the source/destination is characterized by an 8-bit number. The range of possible values extends from 0 to 255
- The access level of the source/destination is also given by an 8-bit number in the range from 0 to 255.
- The optional values network address and MAC address of the source or destination, respectively, are only present, when the source type or destination type have the value 1.
- The network address must be a valid 6-byte network address according to ISO/OSI-rules.
- The MAC address is a string according to the rules for MAC addresses.

Example data structure

```
PROFIBUS_FSPMM2_INITIATE_REQ_T tInitReqData =
{
    .ulRemAdd = 2,
    .usSendTimeout = 200,
    .bFeaturesSupported1 = 0x01,
    .bFeaturesSupported2 = 0,
    .bProfileFeaturesSupported1 = 0,
    .bProfileFeaturesSupported2 = 0,
    .usProfileIdentNumber = 0,
    .tAddAddrParam.bS_Type = 0,
    .tAddAddrParam.bS_Len = 2,
    .tAddAddrParam.bD_Type = 0,
    .tAddAddrParam.bD_Len = 2,
    .tAddAddrParam.abAddParam = {
        0,  /* S_API */
        0,  /* S_SCL */
        0,  /* D_API */
        0,  /* D_SCL */
    },
};
```

3.4.10 Execution of Basic Services

The following figures (*Figure 5: Profibus Master DP V1 Class2 Services* on page 40 and *Figure 6: Profibus Master DP V1 Class2 Close/Abort* on page 41) illustrate the sequence of the steps which are processed when executing the other basic services.

Read Service/ Write Service/ Data Transport Service

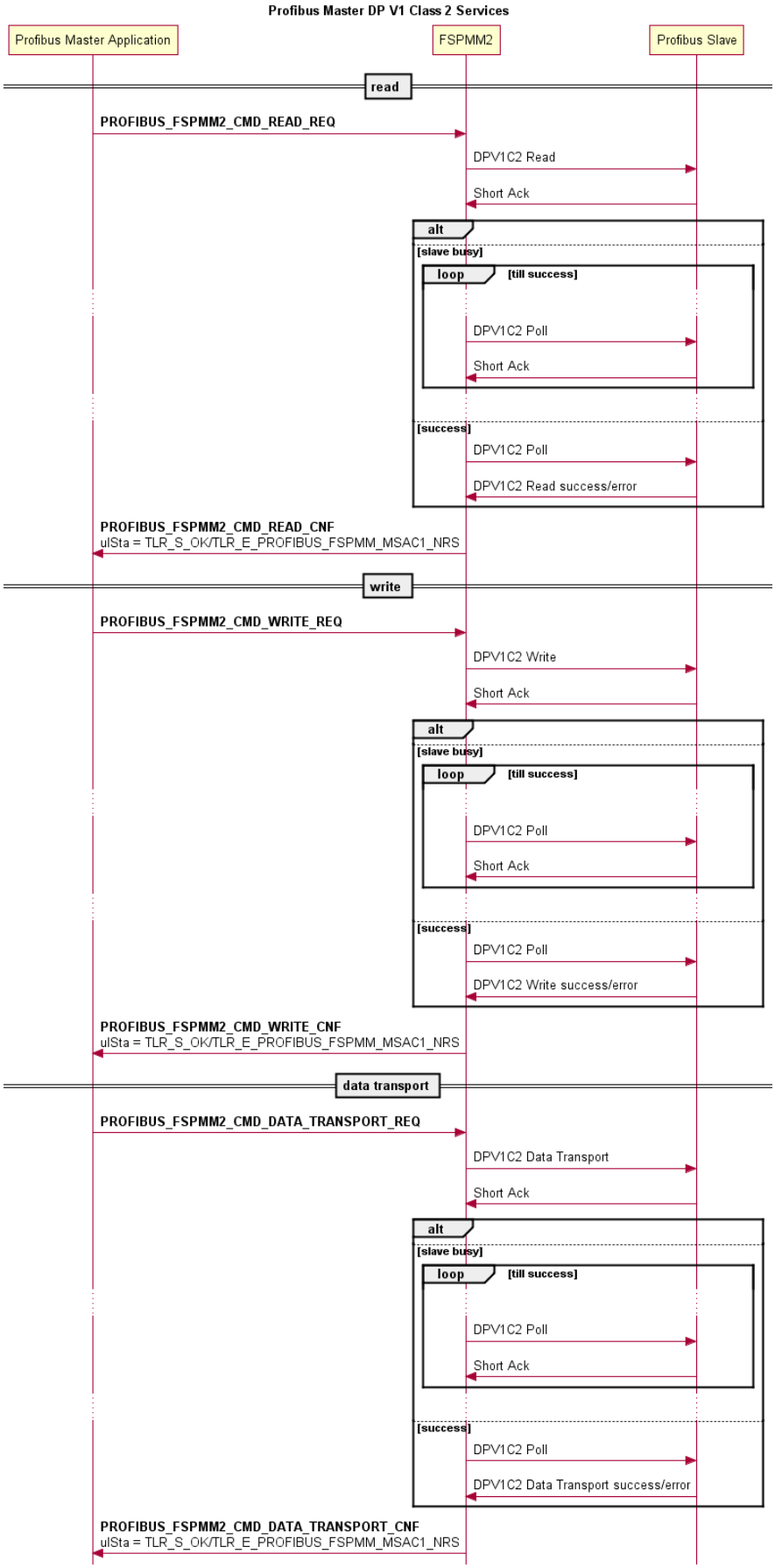

Figure 5: Profibus Master DP V1 Class2 Services

Abort Service / Close Service

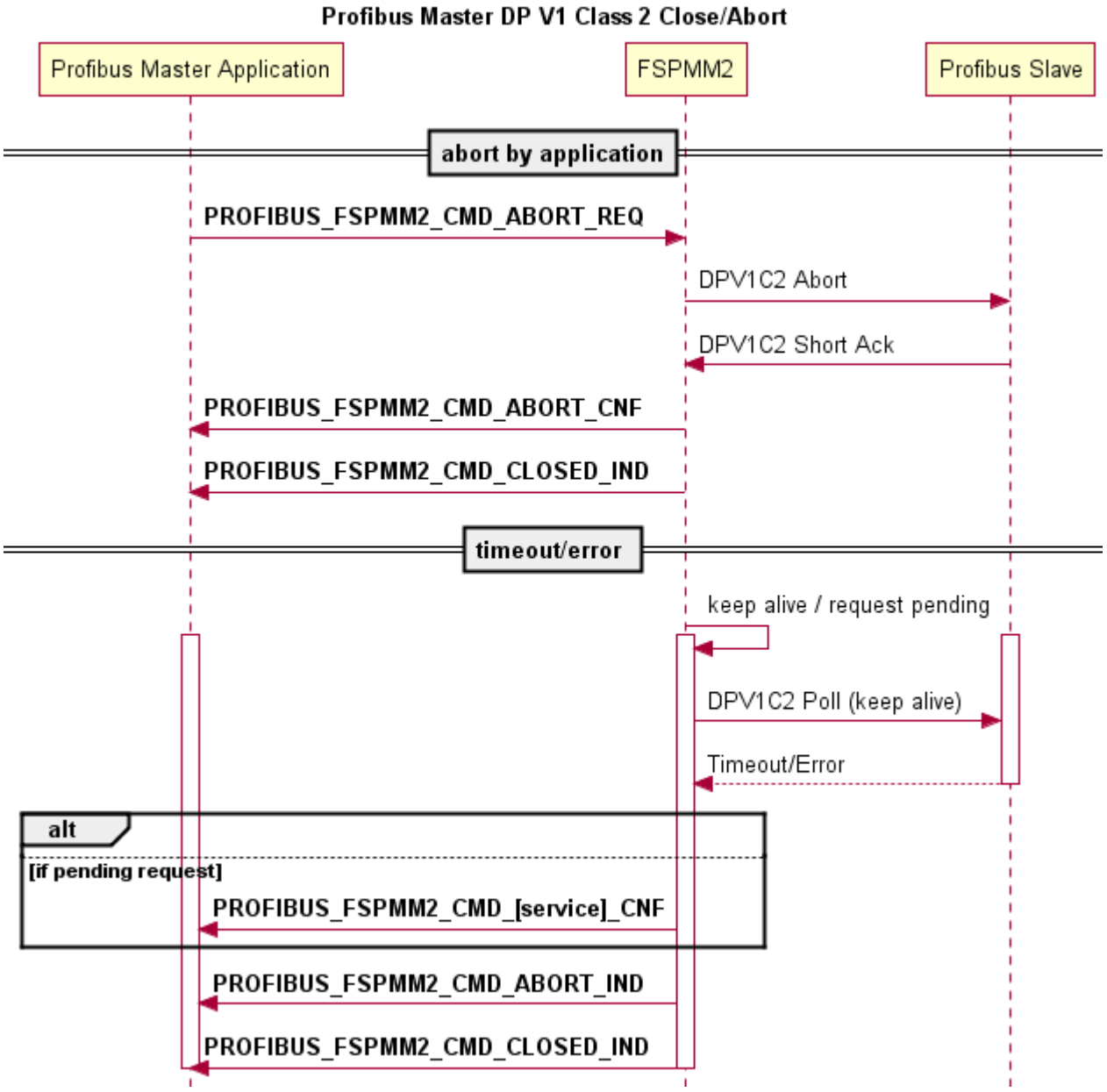

Figure 6: Profibus Master DP V1 Class2 Close/Abort

3.4.11 Requirements for the Class2 Master's User Task

After receiving the positive confirmation packet `PROFIBUS_FSPMM2_CMD_INITIATE_CNF_POS`, the Class2 connection is now in the established, i.e. 'Opened' state. Because the connection is supervised now from this time on and a connection MSAC2M_Abort or a connection MSAC2M_Close may occur at any time, the user application has to look now permanently or at least cyclically if the Host Mailbox of the Hilscher device indicates reception of an `PROFIBUS_FSPMM2_CMD_ABORT_IND` or `PROFIBUS_FSPMM2_CMD_CLOSED_IND` packet or both packets following one after the other.

Because if a connection is aborted automatically through external influence this will be first indicated by a `PROFIBUS_FSPMM2_CMD_ABORT_IND` packet and because the connection is closed then afterwards automatically too, the `PROFIBUS_FSPMM2_CMD_CLOSED_IND` packet will follow.

3.4.12 Connection Supervision by Internal Timers

The once established DP V1 Class2 connection is monitored in order to detect the failure of the communication partner.

The monitoring concept is based on the following elements:

- Various monitoring timers
- The exchange of idle process data units between the communicating partners which retrigger those timers.

There are various timers both in the master Class2 and in the slave (U-, F-Timer I-Timer) for supervision of the various aspects of the connection. The master provides:

- S-Timer (send timer): This timer is used to determine whether the master is idle.
- R-Timer (receive timer): This timer supervises both the slave and the FDL (OSI model layer 2-parts of the master).

The PROFIBUS DP Slave provides:

- U-Timer (user response timer): The user response timer monitors the slave application It is started with passing the service indication to the application and stopped with passing the service response to the application. On expiration of the U-Timer the slave signals to the master that it is idle.
- F-Timer (fetch response timer): This timer supervises the correct function of the master each timer when a response or a slave-idle signal is sent to the master. On expiration of this timer, the connection is aborted.
- I- Timer (indication timer): This timer supervises the correct function of the master in the following way: When the master fetches data, this time is started and running until the master either sends the next request or an idle message. If this timer expires, the connection will be aborted.

The main mechanisms for controlling the DP V1 Class2 connection are:

- At the master, the use of the R-Timer
- At the slave, the use of the I-Timer

3.5 Functionality of the DL Task (Layer 2)

To get the handle of the process queue of the DL-Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII queue name	Description
"PB_DL_QUE”	Name of the DL-Task process queue

Table 122: FSPMM-Task process queue

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the DL-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the DL-Task.

A part of the functionality of PROFIBUS DP is situated within layer 2 of the OSI/ISO layer model of communication in networks. The DL task (DL = Data Link Layer) provides this functionality as an infrastructure to be used by the higher-level functions on layer 7 and higher.

This functionality concentrates mainly on the following 3 topics

- Data transfer services
- Management of Services Access Points (SAPs)
- Management of the system variables of layer 2

The following data transfer services are provided:

- SDA Service (Send data with acknowledge)
- SDN Service (Send data with no acknowledge)
- SRD Service (Send and request data with reply)
- MSRD-Service (Multi-cast send and request data with reply, currently not supported)

The following services are provided for the management of Service Access Points (SAPs)

- SAP Activate
- RSAP Activate
- SAP Deactivate

Furthermore, there are functions providing access to system variables of layer 2

- Set Value
- Set Value Bus Parameter Set

3.5.1 Data Transfer Services

All these services are requested by one participant (denominated as the “initiator”) and are performed by another participant (denominated as the “responder”). Both master and slaves may act as responders, but only masters are authorized to act as an initiator of a data transfer.

All these services connect a local user with a remote user.

In this context, the following definitions apply:

Local user: A local user is a user of layer 2 functionality (FDL) at a master station.

Remote user: A remote user is the FDL user at the remote station (slave or master).

Link service data unit (L_sdu): The data area to be transferred is denominated as link service data unit or shortly as L_sdu.

3.5.1.1 SDA Service

The SDA-Service (SDA = Send data with acknowledge) provides acknowledged connectionless data transfer with immediate response. By the SDA service, a local user gets the possibility to send data (i.e. the L_sdu) to one single remote user (i.e. remote station).

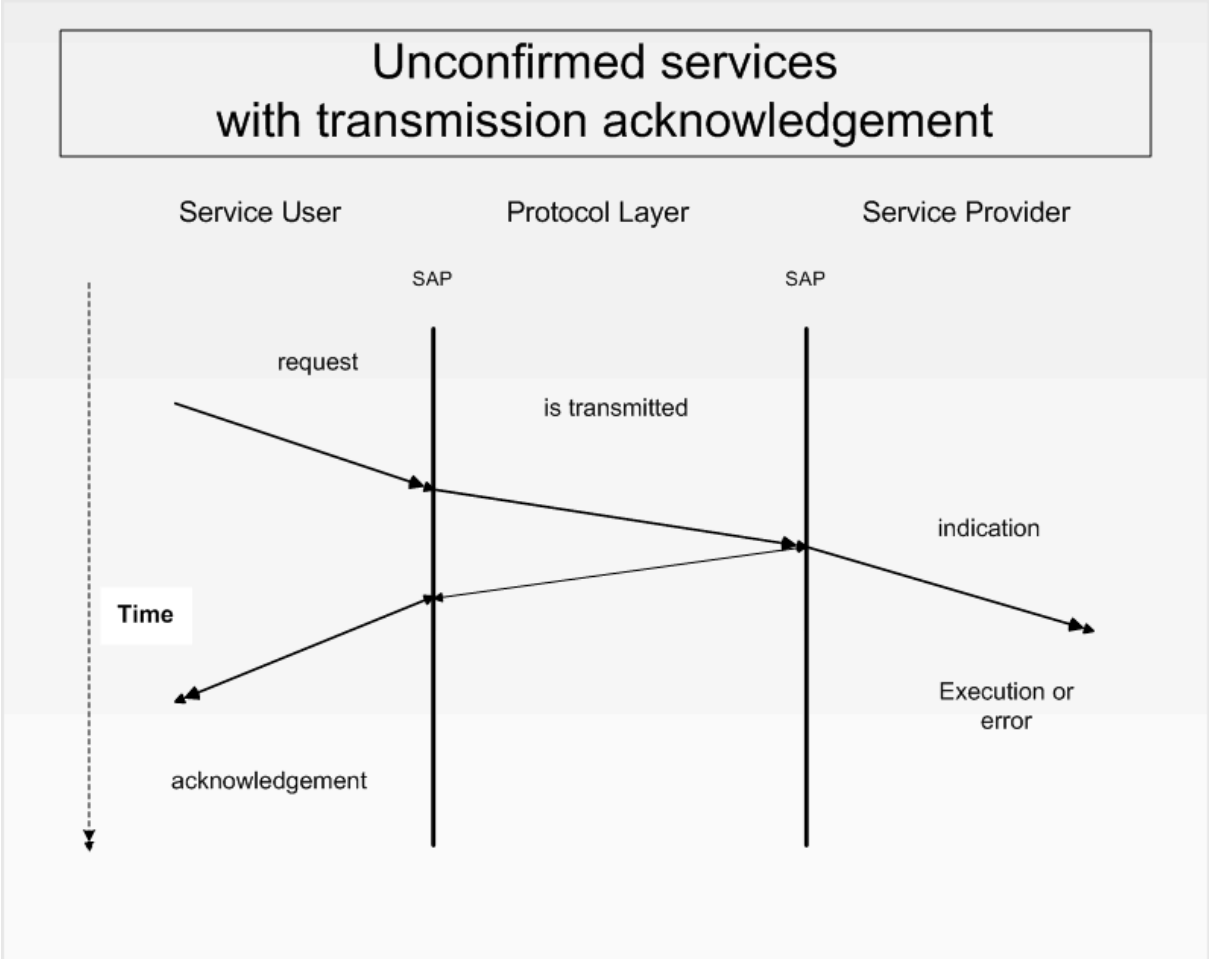

Figure 7: SDA Service with Acknowledgement

The end of the data transfer is acknowledged but not confirmed (which would mean waiting for finishing processing of the data instead of simply receiving the data). This means:

The local user is informed about successful reception or the non-reception of the data by an explicit acknowledgement. However, the user is not informed about successful execution of the request.

In case of an error during data transmission the data transfer will be repeated as often as necessary.

Note: The SDA service is usually not applied in PROFIBUS DP systems (but important for PROFIBUS FMS systems). Nevertheless, it is implemented due to compatibility reasons. This service is mainly used for master-to-master-communication.

The following packets provide the SDA functionality in the PROFIBUS DP master firmware:

- PROFIBUS_DL_CMD_DATA_ACK_REQ/CNF - Send SDA Service
- PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service

3.5.1.2 SDN Service

The SDN -Service (SDN = Send data with no acknowledge) provides unacknowledged connectionless data transfer. By the SDN service, a local user gets the possibility to send data (i.e. the L_sdu) to

- one single remote user (i.e. remote station).
- many remote users (remote stations/ multi-cast communication)
- all remote users/remote stations within the PROFIBUS DP network synchronously

The confirmation of an SDN service only locally confirms that the service has been received for further transmission at the bus, but it neither confirms successful transmission to the receiver nor successful processing there.

In case of an error during data transmission the data transfer will be repeated as often as necessary.

If the remote station receives the link service data unit in an error-free condition, it will be processed (i.e. handled over to the remote user). The local user, however, is not able to obtain information whether processing has taken place as originally intended.

The following packets provide the SDN functionality in the PROFIBUS DP master firmware:

- PROFIBUS_DL_CMD_DATA_REQ/CNF - Send SDN Service
- PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication

3.5.1.3 SRD Service

The SRD-Service (SRD = Send and request data with reply) is designed to provide bidirectional connectionless data exchanged. By the SRD service, a local user gets the possibility to send and synchronously request data (i.e. the L_sdu) to one single remote user (i.e. remote station) at the same time. The requested data must have been stored for fetching there earlier. For the local user, it is also possible to request data without sending data in parallel (pure request command).

In case of successful execution, the local user will receive the requested data. Otherwise, an indication will be sent if the requested data are not available. Both cases confirm the correct reception of the transmitted data at the remote station. However, if the transmitted data have not been received properly, a negative confirmation will be sent.

In case of an error during data transmission, both the data transfer and the request will be repeated as often as necessary.

The following packets provide the SRD functionality in the PROFIBUS DP master firmware:

- PROFIBUS_DL_CMD_DATA_REPLY_REQ/CNF - Send SRD Service
- PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication

3.5.1.4 MSRD Service

An MSRD service request is simply an SRD request where the slave answers with a multi-cast telegram instead of an ordinary telegram for a one-to-one communication relation. This service is not supported by the current firmware.

3.5.2 Management of Services Access Points

The following services are provided concerning the administration of service access points (SAPs):

- Activation of SAP
- Activation of SAP for responder (RSAP)
- Deactivation of SAP

3.5.2.1 Services Access Points

A service access point is a means for managing communication between different communication layers correctly; it is associated with and identified by a non-negative integer numeric value. Service access points are used to characterize different kinds of services within the network. The available numeric values range from 0 to 63. Also the value 64 is permitted, but it represents the NIL SAP.

Note: NIL SAP in this context means that the telegram does not contain any SAP information. Such telegrams are used by PROFIBUS DP for performing cyclic communication.

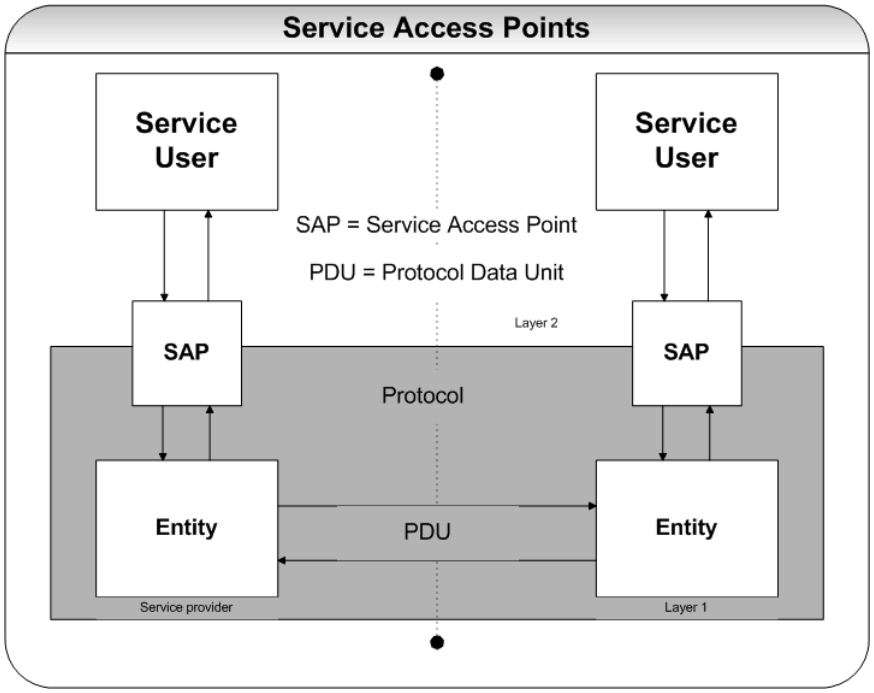

Figure 8: Service Access Points

However, some values might be predefined by other services such as the OSI layer 7 services of PROFIBUS DP. Especially the values 50 (0x32), 51 (0x33), 54 (0x36) and 62 (0x3E) are usually applied for special purposes at the PROFIBUS DP Master.

You can also imagine SAPs as the entry and exit points of a layer in a multi-layered communication model. You may separate among the services for which to define SAPs between user services and management services, so you can introduce a separation between user SAPs (which might be activated by this packet) and previously activated management SAPs.

Along with the SAPs come automatic checking mechanism for rule conformity of the messages and the accompanying transmission-related information stored in the messages. Only if all checks have been passed successfully, the data transfer for which an SAP has been defined will take place, otherwise a message will be issued and no data will be transferred.

Generally, the SAPs should be configured during the start-up phase of the PROFIBUS network.

The philosophy of the SAP concept might look unnecessarily sophisticated, but its real aims justifying this amount of sophistication are:

- Precise Control of messages and transmission channels
- Avoidance of erroneous connections

3.5.2.2 SAP Activate

The following packet provides the SAP activate functionality in the PROFIBUS DP master firmware:

- PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request

This packet is applicable in the following situation:

- It is intended to activate a service access point for an SDA or SDN service
- It is intended to activate a service access point for an SRD or MSRD service with no responder functionality (i.e. the role in service is the one of an initiator).

3.5.2.3 RSAP Activate

The following packet provides the RSAP activate functionality in the PROFIBUS DP master firmware:

- PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ/CNF - Activate an SAP for Responder Functionality (i.e. the role in service is responder or both initiator and responder).

This packet provides the possibility to establish a service access point for an SRD or MSRD service with responder functionality.

3.5.2.4 SAP Deactivate

This service allows the deactivation of a formerly defined service access point, which is not needed any longer. The following packet provides the SAP deactivate functionality in the PROFIBUS DP master firmware:

- PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request

3.5.3 Management of Layer 2 System Variables and Tasks

There are two services available for setting system parameters and variables. One service sets some specific values of the DL Layer. See section 6.3.14 “PROFIBUS_DL_CMD_SET_VALUE_REQ/CNF - Set Value Service”. The other available service loads an entire bus parameter set as a whole. For more information on this service, have a look at section 6.3.12 “PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request”.

There are also some functions dealing with administrative topics such as starting and stopping the DL functionality and stopping the whole DL task:

- PROFIBUS_DL_CMD_START_DLE_REQ/CNF – Starts the DL-Layer
- PROFIBUS_DL_CMD_STOP_DLE_REQ/CNF – Stop DL Layer

3.6 Configuration during running system (Operation State OPERATE)

In process automation it is an important demand that the configuration of a running network can be changed without performing a reset of the devices. Scope of this section is the definition of the configuration during network operation state OPERATE feature for the PROFIBUS DP network. For more information on operation states, see section *PROFIBUS DP Operation Modes* on page 14.

The function ‘Configuration in Run’ (= CiR) can be used when the following requirements are fulfilled:

- PROFIBUS DP Master Firmware/stack V2.3.x.x or higher
- Configuration Software PROFIBUS DP Master DTM in *“Master Settings”* the parameter ‘*Enable configuration download during network state OPERATE*’ needs to be available and activated

However, the following limitations apply:

- CiR is only supported for configuration with database. If no database is available or the device is configured with packets the CiR packets are rejected by the error code `TLR_E_DATABASE_ACCESS_FAILED`.
- Configurations set via the fieldbus system are not considered and the configurations cannot be adjusted.

3.6.1 Configuration Procedure

The configuration procedure is performed according to the following steps:

1. The configuration software PROFIBUS DP Master DTM downloads a new database
2. The Host sends a verify configuration request packet (`RCX_VERIFY_DATABASE_REQ`) to the PROFIBUS DP-Master stack.
3. The firmware compares the new database which has been downloaded in step 1 with the previously active old database and sends the result to the host. The items to be compared comprise bus parameters like e.g. address of the PROFIBUS DP Master, bus speed, and every slave parameter set.
   However, the address of the master and bus speed cannot be changed in the new configuration. If the bus settings are correct, in the next step every slave parameter set will also be checked for possible changes. The checked items include the DPM mapping and other slave parameters. When all slaves have been checked for possible changes the master sends a verify configuration confirmation packet (`RCX_VERIFY_DATABASE_CNF`) containing a complete list of slaves which are
   - new in the configuration,
   - deactivated,
   - changed,
   - unchanged and
   - a list of those slaves whose proposed changes are not possible.

4. The Host receives the confirmation packet for verify configuration (`RCX_VERIFY_DATABASE_CNF`) and has to decide, whether verification was successful, or not
5. If the verification has been successful, then the Host sends the command for activating changes. Otherwise, the remaining steps will not be carried out and an `RCX_FILE_DELETE` request packet has to be sent to the stack.
6. Now, the new configuration is activated by sending the activate database request packet (`RCX_ACTIVATE_DATABASE_REQ`).
   - At first, the master deactivates all slaves having been deleted.
   - Then the master sends new parameter data to the changed slaves. Parameter data is send as it is set at the database. The slave has to decide it can handle the new parameter at runtime or a restart of the slave communication is necessary.
     - A set parameter packet is sent when the parameter data is changed. This is done at state *data exchange*.
     - The address mapping is changed when the address table is changed. This is handled without any influence at the PROFIBUS communication. The host application has to take care about valid I/O data at the dual port memory.
     - If configuration data is changed the slave has to restart the communication. The master will unlock the slave to abort the communication relationship and start the slave again in the same manner as after a power on.
   - Afterwards the PROFIBUS DP Master sends the configuration to all new slaves and sets them to active state regarding to the setting of each slave.
   - Finally, the firmware saves the new database as current database and sends the confirmation packet `RCX_ACTIVATE_DATABASE_CNF` to the host.
7. The Host receives the confirmation packet for activating changes and has to decide, whether the activation was successfully or not.

The following figure illustrates this configuration procedure.

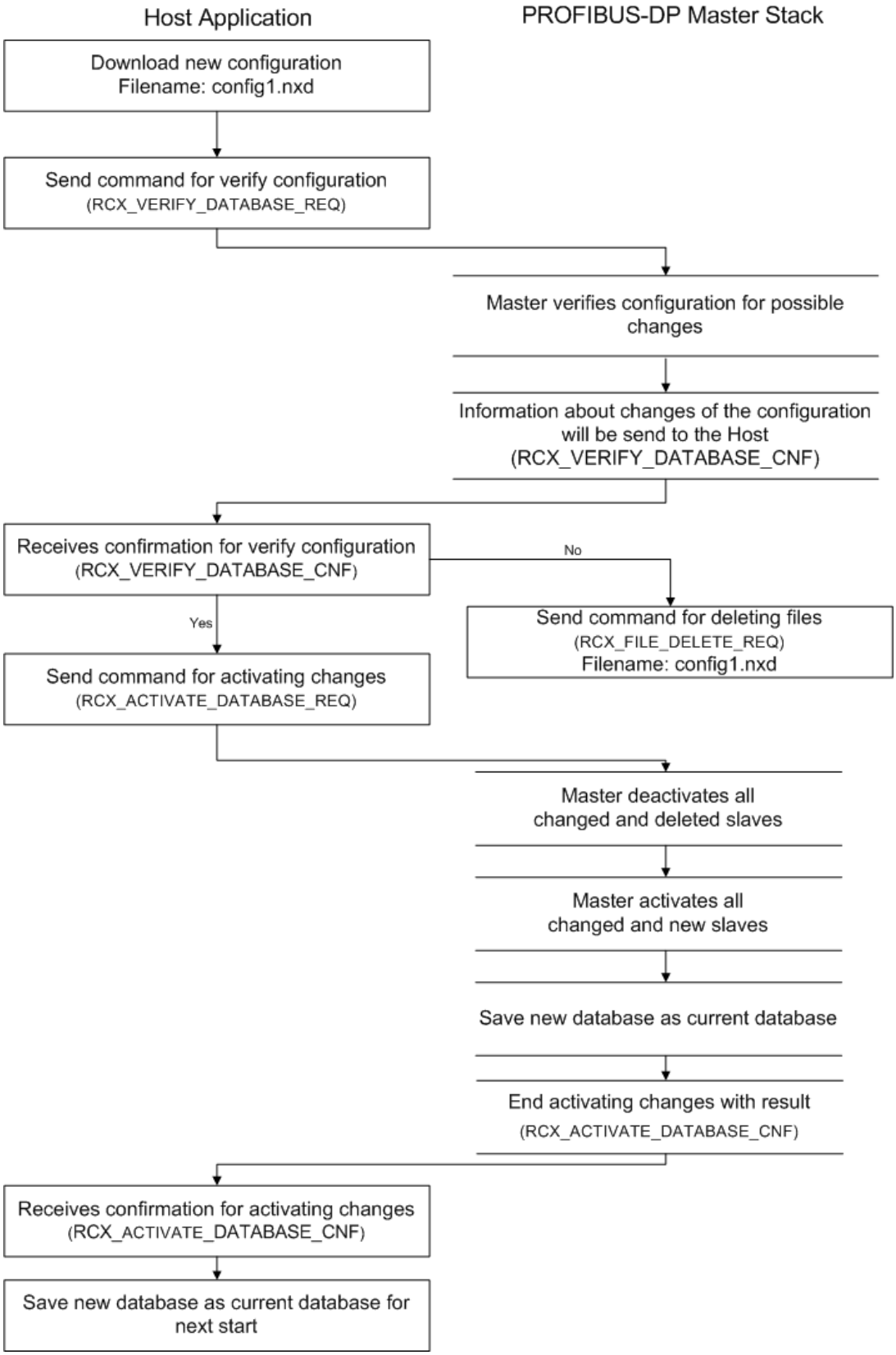

Figure 9: Configuration during Network State OPERATE

3.7 Redundancy Functionality

A redundant master system is defined by

- one primary master device and
- one backup master device.

The primary master is active on the network while the backup master is passive. This is the system, which is prepared for the redundancy case.

An active master will behave like a standard master. Both masters are

- participating in the PROFIBUS ring and passing the PROFIBUS token
- sends the FDL status telegrams

while the passive master

- is not exchanging I/O data

The backup master (passive) will check its LAS (list of active devices) if the primary master exists. In case the primary master disappears from or appears again in the LAS of the backup master, the backup master indicates to the host application about the missing primary master. At this state the host application has to perform a switch over to set the backup master to the active mode.

By performing a switch over the backup master changes the current bus address to the bus address of the primary master and establishes communication to the slave devices.

Note: The redundancy functionality of the PROFIBUS DP Master firmware can only be used, when the host application supports this functionality.

Note: In PNO mode, the redundancy functionality can be used with DPV0 slaves only. DPV1 slaves are currently not supported as they can fall off the bus during the switch over. There is a special Hilscher mode allowing to use DPV1 slaves as well. This mode is compatible to the PNO mode concerning its behavior and the request and indication packets. It can be switched on by setting bit 1 in parameter `bOption` of packet `PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ` to 1.

Note: The host application needs to have access to the dual-port memory of both the active and the passive master.

Note: In the configuration the setting for 'Start of bus communication' has to be set to 'Controlled by application'.

Note: The switch over has to be finished before the 'watchdog control time' in the slave device(s) expires otherwise slave(s) can go off the bus.

The following figure shows the basic functionality, the host application has to support.

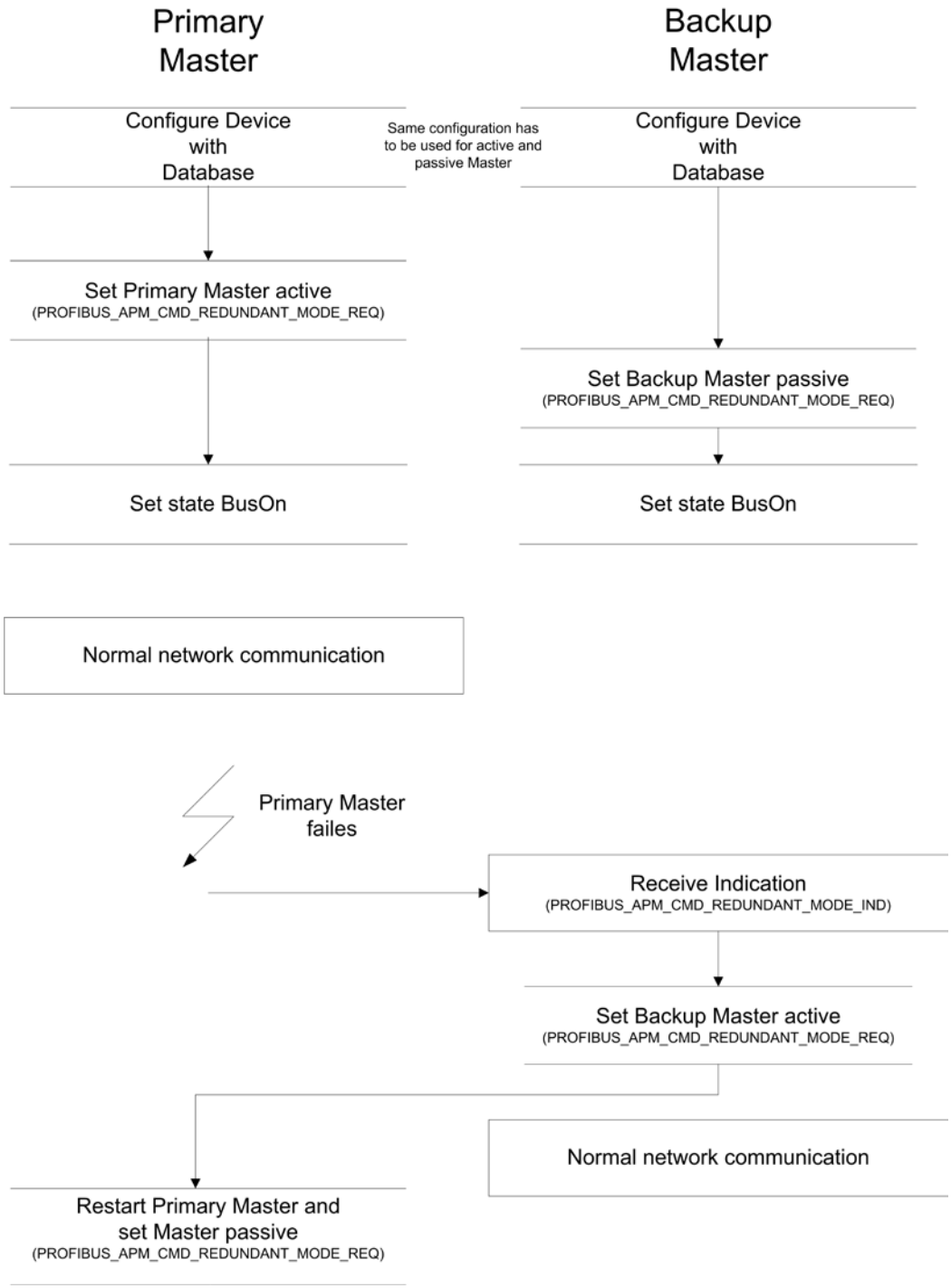

Figure 10: Redundancy Functionality – Host Application

Note: The host application has to transfer the output data from the dual-port memory of the primary master into the dual-port memory of the backup master and to perform a handshake for the output data before performing the final switch over from the primary master to backup master.

4 Configuration of Bus and Slave Parameters

In general, the master can be configured either by

- using SYCON.net configuration software,
- using the API (packets) to set the bus parameters and the slave parameters.

This section explains how to configure the bus parameters and slave parameters of the master via the API using packets.

4.1 Configuring the Master using Packets

In order to set the bus parameters of the master, the *PROFIBUS_FSPMM_CMD_INIT_REQ/CNF – Initialization Command* packet (page 83) has to be sent to the protocol stack.

In order to set the slave parameters and to supply them with warmstart parameters, then *PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode* (page 86) has to be sent to the protocol stack.

4.2 Detailed Description of Bus and Master Parameters

Both the bus and the master need to be configured. The accurate choice of the bus parameters is the foundation of correctly operating data exchange on the PROFIBUS network.

The following table contains relevant information about the bus parameters (including the master's parameters) for the PROFIBUS DP-Master firmware such as a short explanation of the meaning of the parameter and ranges of allowed values:

Bus and Master Parameters

Parameter	Meaning	Range of Value
Bus_Para_Len	Length of the Bus_Para including the field Bus_Para_Len itself	66-103
FDL_Add	Address of the PROFIBUS Master device	1-125
Baud_rate	Transmission speed	0-11
T_{SL}	Slot-time	37-16383
Min T_{SDR}	Min. station delay responder	1-1023
Max T_{SDR}	Max. station delay responder	1-1023
T_{QUI}	Quiet time	0-127
T_{SET}	Setup time	1-255
T_{TR}	Target rotation time	>= 255
G	Gap update factor	1-255
HSA	Highest station address	1-126
Max_Retry_Limit	Retries if error occurs	1-15
Bp_Flag	See separate explanation below	0-255
Min_Slave_Interval	Minimum Slave Interval Time	0-65535
Poll_Timeout	Class2 Poll timeout	0-65535
Data_Control_Time	Data Control Time	1-65535
Alarm_Max	Maximum Alarms	7-32
Max_User_Global_Control	Maximum allowed parallel active USER Global Control Commands	1-255
Reserved	4 reserved octets	
Master_User_Data_Len	Contains the length of the Master_User_Data including the field Master_User_Data_Len itself	2-34
Master_Class2_Name[32]	Name of the Master Class2 (currently not used)	
Master_User_Data[0..32]	USER specific Parameter data – variable size 0 ... 32 configured by Master_User_Data_Len	
T_{CL}	Isochronous cycle time (currently not used)	
Max_T_{SH}	Maximum Shift Time (currently not used)	

Table 22: Bus and Master Parameters

Bus_Para_Len

This parameter denotes the length of the bus parameter structure in bytes including the field itself and is calculated as follows:

32 bytes (fixed part of the packet)

+ 2 bytes length of parameter `Master_User_Data_Len` (which has the range 2 … 34)(1)

+ 0 … 32 bytes used for `Master_User_Data`

+ 32 bytes (`Master_Class2_Name`)

Two optional parameters for DPV2: 4 bytes optionally for T_{CL} and 1 byte Max_T_{SH}

Remark (1): The value of `Master_User_Data_Len` is the length of the `Master_User_Data_Len` parameter (2 bytes) and number of bytes used in the `Master_User_Data` (0 … 32 bytes). The range of values of the `Master_User_Data_Len` parameter is 2 … 34.

Bus_Para_Len is either

- 66 … 98 (= 64 byte + value of parameter `Master_User_Data_Len`)
- 71 … 103 (= 64 byte + value of parameter `Master_User_Data_Len` + 5 bytes of DPV2 parameters)

FDL_Add

This parameter is used for defining the address of the PROFIBUS DP Master itself.

Note:	Configuration of addresses in a PROFIBUS network must be performed in such a way that all addresses are used uniquely. No addresses may appear twice or more often within the PROFIBUS network!

Note:	Do not use the address 0 as the master address although this would both be possible and allowed because this address is often used by configuration and diagnosis devices. Therefore, it is a good idea to choose a non-zero address for the PROFIBUS DP Master.

Baud_rate

This parameter contains the baud rate (i.e. the transmission speed of data signals) used by all stations of the PROFIBUS network in a coded manner. This value must be set to the same value at all slave stations within the network. If this is not the case, all stations with deviating transmission speed setting will not operate correctly. The coding is as follows:

Symbolic Name	Baud rate	Value
PROFIBUS_DL_DATA_RATE_96	9,6 kBit/s	0
PROFIBUS_DL_DATA_RATE_19_2	19,2 kBit/s	1
PROFIBUS_DL_DATA_RATE_93_75	93,75 kBit/s	2
PROFIBUS_DL_DATA_RATE_187_5	187,5 kBit/s	3
PROFIBUS_DL_DATA_RATE_500	500 kBit/s	4
PROFIBUS_DL_DATA_RATE_1500	1500 kBit/s	6
PROFIBUS_DL_DATA_RATE_3000	3000 kBit/s	7
PROFIBUS_DL_DATA_RATE_6000	6000 kBit/s	8
PROFIBUS_DL_DATA_RATE_12000	12000kBits/s	9
PROFIBUS_DL_DATA_RATE_31_25	31.25kBits/s	10
PROFIBUS_DL_DATA_RATE_45_45	45.45 kBits/s	11
PROFIBUS_DL_DATA_RATE_AUTO	Auto-detection mode. This option is currently not supported.	15

Table 23: Supported Baud Rates

T_{SL} (Slot Time)

This parameter defines the 'Wait for receipt' monitoring time, i.e. the time the master will wait for immediate response or confirmation after sending a request. If the slot time has passed and no response arrived, the telegram will be sent again until the maximum retry limit (i.e. maximum allowed number of repetitions of sending the telegram) has been reached.

This time is identical to the time interval within which the master must either send a request telegram or give the token to the next station.

Allowed values for the slot time range from 37 to 16383. The default value however depends on the chosen baud rate.

Min T_{SDR}

This is the shortest time that must elapse before a remote recipient (Responder) may send an acknowledgement of a received query telegram. The shortest time between the reception of the last Bit of a telegram to the sending of the first Bit of a following telegram.

Allowed values for the slot time range from 1 to 65535. Default value is 11.

Max T_{SDR}

This is the longest time that must elapse before a sender (Requestor) is allowed to send a further query telegram. The largest time between reception of the last Bit of a telegram to the sending of the first bit of a following telegram. The sender (Requestor, Master) must wait at least for this time after the sending of an unacknowledged telegram (e.g. Broadcast only) before a new telegram is sent.

Allowed values for the slot time range from 1 to 65535. The default value depends from the baud rate.)

T_{QUI} (Quiet Time)

The quiet time is defined as the time delay that occurs for modulators (Modulator-trip time) and Repeaters (Repeater-switch time) for the change over from sending to receiving.

Allowed values for the slot time range from 0 to 127. (The default value depends from the baud rate.)

T_{SET} (Setup Time)

The setup time represents the minimum period “reaction time” between the receipt of an acknowledgement to the sending of a new query telegram (Reaction) by the Sender (Requestor).

Allowed values for the slot time range from 0 to 255. (The default value depends from the baud rate.)

T_{TR} (Target Rotation Time)

The target rotation time is defined as the pre-set nominal token cycling time within the sender authorization (Token) will cycle around the ring. How much time the Master still has available for sending data telegrams to the Slaves is dependent on the difference between the nominal and the actual token cycling time.

Allowed values for the target rotation time range from 1 .. 2^{24}-1 (=16.777.215). (The default value depends on the number of slaves attached to the master and their module configuration)

G (GAP Actualization Factor)

The GAP Actualization Factor is applied for determining after how many token cycles an added participant is accepted into the token ring. After expiry of the time G*TTR, the station searches to see whether a further participant wishes to be accepted into the logical ring.

Allowed values for the GAP Actualization Factor range from 1 to 100. (The default value is 10.)

HSA (Highest Station Address)

The ‘Highest Station Address’ parameter represents the highest bus address up to which a Master searches for another Master at the bus in order to pass on the Token. On no account, this station address must be smaller than the Master station address.

Allowed values for the highest station address range from 1 to 126.

Max_Retry_Limit

As already mentioned above, this parameter is the maximum allowed number of repetitions in order to send the telegram before sending will be aborted.

Allowed values for the maximum retry limit range from 1 to 15. The default value depends on the baud rate.

Bp_Flag

This flag decides on the system behavior if the master recognizes one slave station not to be responsive any more. If the most significant bit is set, then 'Auto Clear' mode is activated meaning the operational state of the affected slave will be set from OPERATIONAL to STOP by the master. Otherwise, the slave will remain in OPERATIONAL state.

Min_Slave_Interval

The minimum slave interval defines the minimum time period between two slave list cycles. The maximum value that the active Stations require is always given.

Allowed values for the minimum slave interval range from 1.. 2^{16}-1 (=65535). The default value depends from the types of slaves used in the PROFIBUS network.

Poll_Timeout

The Class2 Poll timeout parameter is only relevant when working as DP V1 Class2 master (currently not supported). It defines the maximum time in a Master-Master communication relationship within which the answer must be fetched by the requestor.

Allowed values for the Class2 Poll timeout parameter range from 0... 2^{16}-1 (=65535).

Data_Control_Time

The data control time defines the time within the Data_Transfer_List is updated at least once. After the expiration of this period, the Master (Class1) reports its operating condition automatically via the 'Global_Control' command.

Allowed values for the data control time from 1... 2^{32}-1 (= 4.294.967.295). The default value depends from the baud rate.

Master Setting

This parameter may only have the values 0 or 1. It indicates whether word-oriented IO data (16 bit data) are handled in Motorola (Master Setting = 0) or Intel format (Master Setting = 1, high and low byte are swapped, i.e. exchanged by each other).

Note: The former parameter `Alarm_Max` is no longer supported.

Max_User_Global_Control

This parameter specifies the allowed maximum number of simultaneously active Global Control requests from the user. This is a capability of the PROFIBUS DP master. This value can range from 1 to 255, but it might be reasonable to set it to 16 in order to provide the possibility to issue one SYNC and one FREEZE command for each of the 8 slave groups.

Reserved

This is a reserved area.

Master_User_Data_Len

It specifies the length of the user data area at the master and contains the length of the field itself. It may not exceed 34, see Master_User_Data[32] below.

Master_Class2_Name[32]

The ‘Master Class2 Name’ parameter is only relevant when working as DP V1 Class2 master (currently not supported) but must be provided to access Master_User_Data. It represents the name of a PROFIBUS DP V1 Class2 master, if any is present within the PROFIBUS DP network. It is a field with a length of 32 bytes.

Master_User_Data[0..32]

The first two bytes are spent to set the PROFIBUS Ident Number. Therefore the variable Master_User_Data_Len has to be set minimum to 4. If the values are set to 2 the stack will ignore this Ident number.

```
Master_User_Data[0] = IdentNumberLowByte
Master_User_Data[1] = IdentNumberHighByte
```

The next four bytes are used to overwrite some internal FW/HW revision variables for the stack.

```
Master_User_Data[2] = bHardVrsUsrDdlm
Master_User_Data[3] = bFirmVrsUsrDdlm
Master_User_Data[4] = bHardVrsUsr
Master_User_Data[5] = bFirmVrsUsr
```

This variables are returned with service Get_Master_Diag() of a master - master communication. Usually this values can be set to 0 and the stack will use it internal values. If it is necessary to change this values e.g. to adopt to your own HW revision, it can be overwritten if the particular value is <> 0. There for the variable Master_User_Data_Len has to set minimum to 8.

The next three bytes are reserved and must be set to zero

The next byte is used to reduce the verification of DPv1 Class1 and Class2 read/write response length check.

```
Master_User_Data[9] = 1
```

If set to 1 the read response is tolerated if it contains too much data, the write response is tolerated if it contains data. If set to 0 the default checks are used. To use the feature the variable Master_User_Data_Len has to be set minimum to 12.

T_{CL}

Isochronous cycle time.

Note: This item is currently not supported. It is only relevant in the context of DP V2.

Max_T_{SH}

Maximum shift time.

Note: This item is currently not supported. It is only relevant in the context of DP V2.

Conditions for Bus Parameters

In order to get correct communication on the PROFIBUS network, it is necessary that the following conditions for some important bus parameters are fulfilled:

$T_{QUI} < \min T_{SDR}$

$T_{RDY} < \min T_{SDR}$

$T_{QUI} < T_{RDY}$

4.3 Detailed Description of Slave Parameters

Additionally to the configuration of the bus and the master, also the slaves connected with the master need to be configured.

The following table informs about the relevant slave parameters for the PROFIBUS DP-Master firmware such as an explanation of the meaning of the parameter and ranges of allowed values:

Slave Parameters, Meanings and Ranges

Parameter	Meaning	Range of Values
Slave Address	Slave address of slave to be configured	0…125
Slave_Para_Len	Length of parameter set (i.e. the length of whole data set inclusive the length parameter)	0-65535
Sl_Flag	Slave flags (in accordance to the DP extension to EN 50170(DPV1 support))	0-255
Slave_Type	Slave type (should always be 0 for a DP-Slave)	0-255
Max_Diag_Data_Len	Maximum of diagnostic data length	
Max_Alarm_Len	Maximum length of a Alarm PDU	4-64
Max_Channel_Data_Length	Maximum size of channel date	4-244
Diag_Upd_Delay	Number of requests for a slave diagnostic (PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ) while request for parameters is still set	0-15
Alarm_Mode	Maximum number of possible active alarms	0-7
Add_Sl_Flag	Ignore auto_clear	0-3
C1_Timeout	Timeout for C1 services	0-65535
bReserved[4]	Reserved for further use	
Prm_Data_Len	Length of the following Prm_Data area including the length of the size indicator	9-246
Prm_Data	Parameter data area, see separate description of the structure	
Cfg_Data_Len	Length of the following a Cfg_Data area including the length of the size indicator	3-246
Cfg_Data	Check configuration data area, see separate description of the structure	
Add_Tab_Len	Length of the following Add_Tab data area including the length of the size indicator	2-65504
Add_Tab	Address assignment table, see separate description of the structure	
Slave_User_Data_Len	Length of the following slave user specific data area (Slave_User_Data) including the length of the size indicator	2-65504
Slave_User_Data	User- or manufacturer-specific data area describing the slave for the master	

Table 24:Slave Parameters, their Meanings and their Ranges of allowed Values

Slave Address

This parameter indicates the address of the slave to be configured.

Slave_Para_Len

This parameter indicates the length of the whole data structure containing the slave parameter set given in bytes. This length is computed including the length parameter itself.

It is also used for another purpose: By setting this parameter to zero, a PROFIBUS DP slave parameter set can be deleted easily.

Sl_Flag

This parameter (1 byte) contains some flags, which are explained by the table below:

Explanation of Bits of `Sl_Flag` within Slave Parameter Block			
Bit	**Name**	**Value**	**Description**
D7	Active	0	DP slave is deactivated
		1	DP slave is activated
D6	New_Prm	0	DP slave receives data
		1	DP slave receives new PRM data
D5	Fail_Safe	0	DP slave receives zero data in CLEAR mode
		1	DP slave receives no data in CLEAR mode
D4			Reserved
D3	DPV1_Supported	0	DP slave functionality according to IEC 61158/EN 50170
		1	DP slave functionality according to DPV1
D2	DPV1_Data_Type	0	CFG data interpretation according to IEC 61158/EN 50170
		1	CFG data interpretation according to DPV1 (currently not supported). This would mean additional configuration data according to PROFIBUS **DPV1** extension is used
D1	Extra_Alarm_SAP	0	DPV1-Master acknowledges alarms via SAP 51. The master uses SAP 51 for DPV1 read/write and for alarm acknowledge to this slave.
		1	DPV1-Master acknowledges alarms via SAP 50. The master uses SAP 50 for the alarm acknowledge to this slave. However, the master still uses SAP 51 for DPV1 read/write services. This setting may cause a higher performance because SAP 50 is used exclusively for the alarm acknowledge and cannot be delayed by a running DPV1 read/write service. Using this feature requires that the slave supports it.
D0	Reserved		Reserved

Table 25: Explanation of Bits of Sl_Flag within Slave Parameter Block

Slave_Type

This parameter contains manufacturer-specific information about the type of DP slave.

Value	Meaning
0	DP slave
1 - 15	Reserved by PROFIBUS norm
16 - 255	Manufacturer-specific meaning

Table 26: Meaning of Slave_Type

Max_Diag_Data_Len

This parameter indicates the maximum length of diagnostic data the slaves should send to the master.

Max_Alarm_Len

This parameter specifies the maximum length of the data area for DPV1 alarms. This value must be at least 4 and may not exceed the lower value of

- Max_Diag_Data_Len – 6
- 64.

Max_Channel_Data_Length

This parameter indicates the maximum size of channel data (i.e. the size of the PDU of the status machine MSAC1 described in the IEC 61158/EN 50170 specification.

Diag_Upd_Delay

This parameter is used for counting the number of DDLM_Slave_Diag.con confirmations in the state DIAG2 while Diag_Data.Prm_Req continues to be set (for slaves with reduced performance).

Alarm_Mode

This parameter indicates the maximum number of active alarms that can be handled by the PROFIBUS DP master. The table below explains the coding:

Value	Meaning
0	1 alarm of each type possible
1	2 alarms in total
2	4 alarms in total
3	8 alarms in total
4	12 alarms in total
5	16 alarms in total
6	24 alarms in total
7	32 alarms in total

Table 27: Meaning of Alarm_Mode Parameter

Add_Sl_Flag

This parameter (1 byte) contains some flags, which are explained by the table below:

Explanation of Bits of Add_Sl_Flag within Slave Parameter Block			
Bit	**Name**	**Value**	**Description**
D7			Reserved
D6			Reserved
D5			Reserved
D4			Reserved
D3			Reserved
D2			Reserved
D1	Ignore_Aclr	0	process the auto clear function
		1	ignore the auto clear function
D0	NA_To_Abort	0	no abort if NA occurs
		1	abort if NA occurs

Table 28: Explanation of Bits of Sl_Flag within Slave Parameter Block

C1_Timeout

This parameter indicates a 16-bit value specifying a timeout value for Class1 services.

bReserved[4]

This represents a reserved area.

Prm_Data_Len

This parameter contains the length of the subsequent parameter area `Prm_Data` including 2 bytes the length parameter itself. The length is specified in bytes

Prm_Data

This parameter contains the parameter data block.

During its start-up procedure, a slave station will receive such a parameter block when the PROFIBUS DP command 'Set_Prm' is performed.

The parameter block consists

- 7 bytes (also called octets in this context) of norm specific parameters
- and a `User_prm_data` field for storing extended user specific data.

The length of these together must not exceed 244 bytes, however, it is recommended not to exceed an amount of 32 bytes otherwise some restrictions apply, see IEC 61158/EN 50170 specification.

The 7 octets are structured as follows:

Octet 1: Station Status_1

Octet 1 of Prm_Data Slave Parameter			
Bit	**Name**	**Value**	**Description**
D7	Lock_Req		The lock request bit together with the unlock request bit governs how the DP slave will be locked or unlocked, see table below.
D6	Unlock_Req		The unlock request bit together with the lock request bit governs how the DP slave will be locked or unlocked, see table below.
D5	Sync_Req.		Forces operation in SYNC mode if supported by the slave and delivered by the Global_Control function.
D4	Freeze_Req		Forces operation in FREEZE mode if supported by the slave and delivered by the Global_Control function.
D3	WD_On	Watchdog control	
		0	Watchdog control is deactivated
		1	Watchdog control is activated
D2-D0	Reserved		Reserved for future extensions

Table 29: Octet 1 of Prm_Data Slave Parameter

The possible combinations of the `Lock_Req` bit and the `Unlock_Req` bit have the following meaning:

Lock_Req bit	Unlock_Req bit	Meaning
0	0	It is only possible to change the parameter T_{SDR}, all other parameters cannot be changed.
0	1	The DP slave will be unlocked for accesses by other masters.
1	0	The DP slave is locked for accesses from other masters. All parameters are accepted, except minT_{SDR} will be set to 0.
1	1	The DP slave is unlocked for accesses by other masters.

Table 30: Meaning of Combinations of Lock_Req and Unlock_Req Bits

Octets 2 and 3: WD_Fact1 and WD_Fact2

These octets may have values between 0 and 255. Both values represent factors for setting the watchdog control time T_{WD}. If the master fails and subsequently the chosen watchdog time expires, the output data will fall into the safe state.

The watchdog time can be computed in units of multiples of 10 milliseconds (= 0,01 seconds) by simply multiplying `WD_Fact1` with `WD_Fact2`. Thus, values between 10 milliseconds and approximately 650 seconds are selectable for the watchdog time.

Note: Watchdog control is switched on and off using bit D3 of octet 1. Refer to above.

Octet 4: Minimum Station Delay Responder (minT_{SDR})

This is the minimum time a DP slave will wait until it will send the response frame to the master. In case of a pure DP system, this is a value in the range between 0 and the value maxT_{SDR}

In mixed systems, the upper limit is 255 times T_{Bit}.

Octet 5_6: Ident_Number

In these bytes, the slave station reports its identification number, which has been assigned to it by the manufacturer.

Octet 7: Group Ident_Number

This byte may be used as a special identifier for setting up groups.

Octet 8_32: User_Prm_Data DP-Slave specific parameters

This is an extended diagnostic buffer. The values are determined by the slave station and should be described in the manual of the slave station.

Cfg_Data_Len

This parameter contains the length of the subsequent parameter area `Cfg_Data` including 2 bytes the length parameter itself. The length is specified in bytes

Cfg_Data

This parameter contains the check configuration data block.

The parameter block contains configuration data for the slave, which decides on the number of input and outputs of the slave. This data is sent to the slave with the PROFIBUS DP command 'Check_Cfg' to force the slave to compare this configuration with its own internally saved one.

In detail, it is possible to specify here:

- Input data and their size
- Output data and their size
- Manufacturer-specific data and their size
- The amount of consistency to apply

The parameter block consists of

- An identifier byte (either in the general or in the special identifier format, see below)
- A length byte

The length of the complete check configuration data block must not exceed 244 bytes, however, it is recommended not to exceed an amount of 32 bytes otherwise some restrictions apply, see the IEC 61158 or EN 50170 specification.

Add_Tab_Len

This parameter contains the length of the subsequent parameter area `Add_Tab` including 2 bytes the length parameter itself. The length is specified in bytes

Add_Tab

The abAddTab field is a Hilscher-specific field, which configures the different offset addresses of process data within the dual-port memory for modular and simple I/O slaves in common. See the following structure:

abAddTab structure

Variable name	Type	Explanation
Input_Count	byte	number of input offsets following in the IO_Offset table
Output_count	byte	number of output offsets following in the IO_Offset table
IO_Offsets[...]	word array	word or byte IO_Offset in the order: first all input offsets then all output offsets in case of modular stations

Table 31: abAddTab Structure

One module entry in the Cfg_list must result in a corresponding entry in the abAddTab, except modules with a data length of 0. In this table, the offset address within the dual-port memory of each module is stored.

Offset addresses to be specified here are relative to the process image area and begin with 0. If there are multiple offset addresses specified, each new one is again relative to the beginning of the (input or output) process image area.

If the upper bit 15 in the IO_Offset[...] value is set to logical '1' then the address is interpreted by the device as byte offset address, otherwise it is interpreted as word offset address.

If the first specified entry contains an offset of 2 and the second specified entry contains an offset of 0 then the following will apply:

> If its length does not exceed 2 bytes, the second specified entry will be located as the first entry within the process image (at offset 0) and the first specified entry will be located as the second entry within the process image (at offset 2).

Example

A configuration table for the above mentioned configuration with one input module (width 1 byte) and two output modules with a width of 8 and 2 bytes:

- Cfg_Data_Len = 0x05
- Cfg_Data = [0x10, 0x27, 0x21]
- Add_Tab_Len = 0x0A
- Input_Count = 0x01
- Output_Count = 0x02
- IO_Offsets[...] = [0x8000, 0x8002, 0x8000]

In this case, input and output data are arranged as displayed in the following figure:

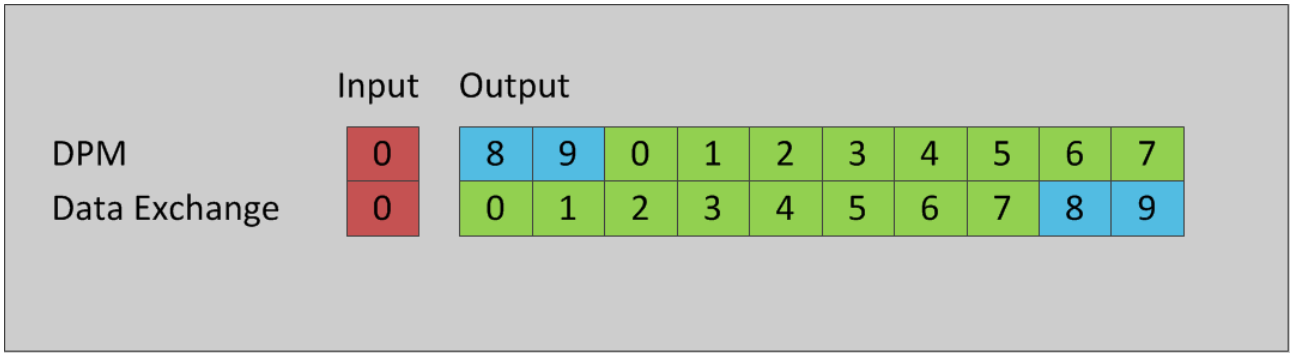

Figure 11: Arrangement of Input and Output Data

Slave_User_Data_Len

This parameter contains the length of the subsequent parameter area `Slave_User_Data` including 2 bytes the length parameter itself. The length is specified in bytes.

Slave_User_Data

This parameter contains slave specific data; see the slaves' documentation for an explanation of the meaning.

5 Status Information

The PROFIBUS Master provides status information in the dual-port memory. The status information has a common block (protocol-independent) and a PROFIBUS-Master-specific block (extended status).

5.1 Common Status

For a description of the common status block, see reference [1].

5.2 Extended Status Block

The Extended Status Block contains

- the Extended Status PROFIBUS DP Master, and
- the Extended Status Field PROFIBUS DP Master.

Extended Status Block Structure

```
typedef struct NETX_EXTENDED_STATE_FIELD_DEFINITION_Ttag
{
  UINT8    abExtendedStatus[172];    /* Default, protocol specific inform. area */
  NETX_EXTENDED_STATE_FIELD_T tExtStateField;  /* Extended status structures */
} NETX_EXTENDED_STATE_FIELD_DEFINITION_T;
```

Offset	Type	Name	Description
0x0050	UINT8[]	abExtendedStatus[172]	Area containing PROFIBUS-DP-related information. See Table 33 on page 72. Offests 0x0090 to 0x00FB are reserved.
0x00FC	NETX_EXTENDED_STATE_FIELD_T	tExtStateField	Structure to define Status Fields and their Properties. Status type and properties are specific to protocol implementation

Table 32: Extended Status Block

Note: Each offset is always related to the begin of corresponding channel start.

Structure PROFIBUS_APM_EXT_STATUS_T contains information about network status (i.e. flags, error counters, events etc.).

Structure NETX_EXTENDED_STATE_FIELD_T (begining at offset 0x00FC) provides the definition of the up to 32 independent State Fields. These state fields can be defined to represent a kind of bit-list, byte-list etc. with up to 65535 entities. In this way a common access mechanism for different state definitions and quantities can be provided independent of protocol implementation.

5.2.1 Extended Status PROFIBUS DP Master

Note: The Extended Status PROFIBUS DP Master is deprecated. The bitlists with status information about configured slaves, slaves in data ecxchange, and slaves with diagnostic are available togehter with the input data. Section *Extended Status Field PROFIBUS DP Master* on page 76 describes the location of the offset addresses in the input data area for these bitlists.

The Extended Status for PROFIBUS DP-Master is structured as follows:

```
typedef struct PROFIBUS_APM_GLOBAL_STATE_FIELD_Ttag {

  /* bit field to show bus and master main errors */
  unsigned char bGlobalBits;
  #define MSK_PROFIBUS_APM_EXT_STA_CTRL_ERR 0x01
  #define MSK_PROFIBUS_APM_EXT_STA_ACLR_ERR 0x02
  #define MSK_PROFIBUS_APM_EXT_STA_NEXC_ERR 0x04
  #define MSK_PROFIBUS_APM_EXT_STA_FATL_ERR 0x08
  #define MSK_PROFIBUS_APM_EXT_STA_NRDY     0x10
  #define MSK_PROFIBUS_APM_EXT_STA_TOUT     0x20


  /* master main states */
  unsigned char   bDPM_state;
  /* USIF_OFFLINE 0x00 */
  /* USIF_STOP    0x40 */
  /* USIF_CLEAR   0x80 */
  /* USIF_OPERATE 0xC0 */

  /* location of error and error code */
 struct
  {
    unsigned char bErr_Rem_Adr; /* 0-125, 255 */
    unsigned char bErr_Event;

  } tError;

  /* counter for the bus error events */
  unsigned short  usBus_Error_Cnt;

  /* counter for  bus timeouts */
  unsigned short  usTime_Out_Cnt;

  /* reserved area */
  unsigned char   abReserved[8];

  /* Bit-Ready, Cfg-Ready and diagnostic display of slave devices */
  unsigned char   abSl_cfg  [16];  /* Slave configuration area */
  unsigned char   abSl_state[16];  /* Slave state information area */
  unsigned char   abSl_diag [16];  /* Slave diagnostic area */
} PROFIBUS_APM_GLOBAL_STATE_FIELD_T;

typedef struct PROFIBUS_APM_EXT_STATUS_Ttag {

  PROFIBUS_APM_GLOBAL_STATE_FIELD_T tGlobStateFiled;

} PROFIBUS_APM_EXT_STATUS_T;
```

Extended Status for PROFIBUS DP Master			
Address	**Type**	**Name**	**Description**
0x50	unsigned char	bGlobalBits	Global bits (Bit field to show bus and master main errors)
0x51	unsigned char	bDPM_state	Master main state (see below)
0x52	unsigned char	bErr_Rem_Adr	Error address (0-125, 255), reserved, not used.
0x53	unsigned char	bErr_Event	Error event, reserved, not used.
0x54	unsigned short	usBus_Error_Cnt	Counter for the bus error events
0x56	unsigned short	usTime_Out_Cnt	Counter for bus timeouts
0x58	unsigned char	abReserved[8]	Reserved
0x60			Bit-Ready, Cfg-Ready and diagnostic display of slave devices:
0x60	unsigned char	abSl_cfg[16]	Slave configuration area
0x70	unsigned char	abSl_state[16]	Slave state information area
0x80	unsigned char	abSl_diag[16]	Slave diagnostic area

Table 33: Extended Status for PROFIBUS DP-Master

bGlobalBits / Global Bits

The global bits byte `bGlobalBits` is a bit field to indicate bus and master main errors containing the following information:

Bit	Meaning
Bit 0	`MSK_PROFIBUS_APM_EXT_STA_CTRL_ERR` CONTROL-ERROR: This error is caused by incorrect parameterization.
Bit 1	`MSK_PROFIBUS_APM_EXT_STA_ACLR_ERR` AUTO-CLEAR-ERROR: The device stopped the communication to all slaves and reached the auto-clear end state
Bit 2	`MSK_PROFIBUS_APM_EXT_STA_NEXC_ERR` NON-EXCHANGE-ERROR: At least one slave has not reached the data exchange state and no process data are exchanged with it.
Bit 3	`MSK_PROFIBUS_APM_EXT_STA_FATL_ERR` (not supported)
Bit 4	`MSK_PROFIBUS_APM_EXT_STA_NRDY` Host-NOT-READY-NOTIFICATION: Indicates if the host program has set its state to operative or not. If the bit is set the host program is not ready to communicate
Bit 5	`MSK_PROFIBUS_APM_EXT_STA_TOUT` TIMEOUT-ERROR: The Device has detected an overstepped timeout supervision time because of rejected PROFIBUS telegrams. It's an indication for bus short circuits while the master interrupts the communication. The number of detected timeouts are fixed in the Time_out_cnt variable. The bit will be set when the first timeout was detected and will not be deleted any more.
Bit 6	Unused
Bit 7	Unused

Table 34: Global Bits Variable

bDPM_state / Master main state

The master main state represents the operation mode of the PROFIBUS DP master stack. This operation mode is defined in section *PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode* on page 86. Allowed operation modes are:

Operation mode	Value
USIF_OFFLINE	0x00
USIF_STOP	0x40
USIF_CLEAR	0x80
USIF_OPERATE	0xC0

Table 35: Operation Modes of the PROFIBUS DP Master and their Values

Changes of the master main state are indicated by the PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication. If the application needs to change this state, it can accomplish this by sending a PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation packet to the FSPMM task, also see section PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation of this document.

bErr_Rem_Adr / Location of error

Reserved, not used.

bErr_Event / Error code

Reserved, not used.

usBus_Error_Cnt / Counter for the bus error events

This variable is a counter for severe bus errors, for example short circuits on the bus.

usTime_Out_Cnt / Counter for bus timeouts

This variable is a counter for the number of rejected PROFIBUS telegrams because of heavy bus error.

abReserved[8] / Reserved

This data block is reserved.

abSl_cfg[16] / List of configured slaves

This variable is a field of 16 bytes and contains the parameterization state of each slave station. The following table shows, which bit is related to which slave station address:

List of configured slaves	D7	D6	D5	D4	D3	D2	D1	D0
abSl_cfg[0]	7	6	5	4	3	2	1	0
abSl_cfg[1]	15	14	13	12	11	10	9	8
abSl_cfg[2]	23	22	21	20	19	18	17	16
…								
abSl_cfg[15]	127	126	125	124	123	122	121	120

Table 36: Relationship between Slave Station Address and the corresponding abSl_cfg Bit

If the abSl_cfg bit of the corresponding slave is logically

- '1', the slave is configured in the master, and serviced in its states.
- '0', the slave is not configured in the master.

abSl_state[16] / List of slaves in data exchange

This variable is a field of 16 bytes and contains the state of each slave station. The following table shows, which bit is related to which slave station address:

List of slaves in data exchange	D7	D6	D5	D4	D3	D2	D1	D0
abSl_state[0]	7	6	5	4	3	2	1	0
abSl_state[1]	15	14	13	12	11	10	9	8
abSl_state[2]	23	22	21	20	19	18	17	16
…								
abSl_state[15]	127	126	125	124	123	122	121	120

Table 37: Relationship between Slave Station Address and the corresponding abSl_state Bit

If the abSl_state bit of the corresponding slave station is logically

- '1', the slave and the master are exchanging their I/O data.
- '0', the slave and the master are not exchanging their I/O datwa.

abSl_diag[16] / List of slaves with diagnostic

This variable is a field of 16 bytes containing the diagnostic bit of each slave. The following table shows the relationship between the slave station address and the corresponding bit in the variable abSl_diag.

List of slaves with diagnostic	D7	D6	D5	D4	D3	D2	D1	D0
abSl_diag[0]	7	6	5	4	3	2	1	0
abSl_diag[1]	15	14	13	12	11	10	9	8
abSl_diag[2]	23	22	21	20	19	18	17	16
…								
abSl_diag[15]	127	126	125	124	123	122	121	120

Table 38: Relationship between Slave Station Address and the corresponding abSl_diag Bit

If the abSl_diag bit of the corresponding slave is logically

- '1', latest received slave diagnostic data are available in the internal diagnostic buffer. This data can be read by the user with a message which is described in the section *PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic* in this document.
- '0', since the last diagnostic buffer read access of the host, no values have been changed in this buffer.

The values in variable abSl_diag are only valid, if the master runs in the main state *OPERATE*.

abSl_state	abSl_state = 0	abSl_state = 1
abSl_diag = 0	No DataIOExchange between master and slave. This slave is not configured.	Slave is present on the bus. DataIOExchange between master and slave.
abSl_diag = 1	The master and the corresponding slave do not exchange their I/O data. The master holds newly received diagnostic data in the internal diagnostic buffer.	Slave is present on the bus. The master and the corresponding slave exchange their I/O data. The master holds newly received diagnostic data in the internal diagnostic buffer.

Table 39: Relationship between abSl_state and abSl_diag bits

5.2.2 Extended Status Field PROFIBUS DP Master

The PROFIBUS DP Master provides 48 bytes for status information:

- 16 byte for list of configured slaves,
- 16 byte for list of slaves in data exchange and
- 16 byte for list of slaves with diagnostic.

This status information is "added" to the input data, thus available within the input data memory and is located "after" the input data. The following table lists the bitlists with these status information and documents the location of the offset address for each bitlist which points to the input data.

Offset	Type	Name	Description/Value
0x00FC	unsigned char	bReserved[3]	Reserved. Do not use.
0x00FF	unsigned char	bNumStateStructs	Number of State Structures defined below = 4
↓	NETX_EXTENDED_ STATE_STRUCT_T	atStateStruct[0]	Structure to define State field properties
0x0100	unsigned char	bStateArea	= 0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	= 1. Corresponds to a bit list (one bit per slave) of configured slaves.
	unsigned short	usNumOfStateEntries	= 128. Corresponds to 128 bits, each representing a slave.
	unsigned long	ulStateOffset	Contains an offset addess to a state field inside input data area 0, which contains the slave configuration area. Table 36 describes the relation between the bitlist and the slave station.
↓	NETX_EXTENDED_ STATE_STRUCT_T	atStateStruct[1]	Structure to define State field properties
0x0108	unsigned char	bStateArea	= 0. State field is located in standard input area of channel 0.
	unsigned char	bStateTypeID	= 2. Corresponds to a bit list (one bit per slave) of slaves in data exchange.
	unsigned short	usNumOfStateEntries	= 128. Corresponds to 128 bits, each representing a slave.
	unsigned long	ulStateOffset	Contains an offset pointer to a state field inside input data area 0, which contains the slave state information area. Table Table 37 describes the relation between the bitlist and the slave station.
↓	NETX_EXTENDED_ STATE_STRUCT_T	atStateStruct[2]	Structure to define State field properties
0x0110	unsigned char	bStateArea	= 0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	= 3. Corresponds to a bit list (one bit per slave) of slaves with diagnostic
	unsigned short	usNumOfStateEntries	= 128. Corresponds to 128 bits, each representing a slave
	unsigned long	ulStateOffset	Contains an offset pointer to a state field inside input data area 0, which contains the slave diagnostic area. Table 38 describes the relation between the bitlist and the slave station.

Offset	Type	Name	Description/Value
↓	NETX_EXTENDED_ STATE_STRUCT_T	atStateStruct[3]	Structure to define State field properties
0x0130	unsigned char	bStateArea	= 0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	= 5. Corresponds to a bit list (one bit per slave) of nodes with a change of the slaves inputs since last DPM update.
	unsigned short	usNumOfStateEntries	= 128. Corresponds to 128 bits, each representing a slave
	unsigned long	ulStateOffset	Contains an offset pointer to a state field inside input data area 0, which contains the area with the slave input change information.

Table 40: Extended Status Field

6 The Application Interface

This chapter defines the application interface of the DP-Master Stack.

The application itself has to be developed as a Task according to the Hilscher's Task Layer Reference Model. The Application-Task is named AP-Task in the following sections and chapters.

The AP-Task's process queue is keeping track of all its incoming packets. It provides the communication channel for the underlying DP-Master Stack. Once, the DP-Master Stack communication is established, events received by the stack are mapped to packets that are sent to the AP-Task's process queue. On one hand every packet has to be evaluated in the AP-Task's context and corresponding actions be executed. On the other hand, Initiator-Services that are be requested by the AP-Task itself are sent via predefined queue macros to the underlying DP-Master Stack queues via packets as well.

The following chapters are describing the packets that may be received or may be sent by the AP-Task.

6.1 The FSPMM-Task

Within the DP-Master Stack, the FSPMM-Task coordinates the underlying DP-Master state machines used for processing of the various services. It consists of the MSCY1M, the MSAC1M, MSAC2M and DMPMM state machines defined in the Chapter 6.3 of the 61158-6 © IEC:2003(E).

Furthermore, it is responsible for all application interactions and represents the counterpart of the AP-Task within the existent DP-Master Stack implementation.

To get the handle of the process queue of the FSPMM-Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII queue name	Description
"FSPMM_QUE"	Name of the FSPMM-Task process queue

Table 41: FSPMM-Task process queue

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the FSPMM-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the FSPMM-Task.

In detail, the following functionality is provided by the FSPMM-Task:

Overview over Packets of the FSPMM -Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.1.1	PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register	0x2224/ 0x2225	81
6.1.2	PROFIBUS_FSPMM_CMD_INIT_REQ/CNF – Initialization Command	0x2200/ 0x2201	83
6.1.3	PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode	0x2206/ 0x2207	86
6.1.4	PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication	0x2214	89
6.1.5	PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic	0x220A/ 0x220B	91
6.1.6	PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic	0x2216	95
6.1.7	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output Data	0x220C/ 0x220D	96
6.1.8	PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input Data	0x220E/ 0x220F	98
6.1.9	PROFIBUS_FSPMM_CMD_READ_REQ/CNF – DP V1 Class1 Read Request	0x2210/ 0x2211	101
6.1.10	PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – DP V1 Class1 Write Request	0x2212/ 0x2213	105
6.1.11	PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification	0x221A	110
6.1.12	PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge	0x221C/ 0x221D	113
6.1.13	PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ/CNF – Download Slave Parameter Set	0x221E/ 0x221F	118

Overview over Packets of the FSPMM -Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.1.14	PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF – Global Control Message	0x2220/ 0x2221	125
6.1.15	PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ/CNF – Activate/Deactivate Slaves	0x2226/ 0x2227	129
6.1.16	PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a Fatal Fault	0x2222	131
6.1.17	PROFIBUS_MSCS1M_CMD_INIT_CS_REQ/CNF – Initialize ClockSync Feature	0x222C/ 0x222D	132
6.1.18	PROFIBUS_MSCS1M_CMD_SET_TIME_REQ – Set Time for *TimeEvent ClockValue* Sequence	0x222E/ 0x222F	134
6.1.19	PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_REQ/CNF – Switch Redundancy	0x2230/ 0x2231	137
6.1.20	PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_REQ/CNF - Input Change of Slaves	0x2232/ 0x2233	140
6.1.21	PROFIBUS_FSPMM_CMD_UPLOAD_REQ/CNF – Upload Slave Parameter Set	0x2234/ 0x2235	142

Table 42: Overview over the Packets of the FSPMM-Master -Task of the PROFIBUS DP-Master Protocol Stack

6.1.1 PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register

This service shall be sent from the AP-Task as the very first command before interacting with the stack. With this command, the stack gets a reference to an existing application and knows where to send unsolicited commands e.g. alarm or diagnostic indications.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_PACKET_APP_REG_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_FSPMM_PACKET_APP_REG_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_APP_REG_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle of the FSPMM-Task process queue
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle of AP-Task process queue
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	sizeof(PROFIBUS_FSPMM_APP_REG_REQ_T)- Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32	0	See *Table 44: PROFIBUS_FSPMM_APP_REG_REQ– Packet Status/Error*
ulCmd	UINT32	0x2224	`PROFIBUS_FSPMS_CMD_APP_REG_REQ` - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	×	Routing, do not change

Table 43: PROFIBUS_FSPMM_APP_REG_REQ – Application Register Request

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok

Table 44: PROFIBUS_FSPMM_APP_REG_REQ– Packet Status/Error

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_PACKET_APP_REG_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_FSPMM_PACKET_APP_REG_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_APP_REG_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See *Table 46: PROFIBUS_FSPMM_APP_REG_CNF - Packet Status/Error*
ulCmd	UINT32	0x2225	PROFIBUS_FSPMM_CMD_APP_REG_CNF- Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 45: PROFIBUS_FSPMM_APP_REG_CNF – Application Register Confirmation

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok

Table 46: PROFIBUS_FSPMM_APP_REG_CNF - Packet Status/Error

6.1.2 PROFIBUS_FSPMM_CMD_INIT_REQ/CNF – Initialization Command

This service needs to be send from the AP-Task in order to initialize of the stack. For this purpose, a complete bus parameter set needs to be transferred successfully. Successful initialization is also a necessary condition for the internal state machine of PROFIBUS DP Master to reach any other state than ‘Offline’. For a detailed description of all members of the bus parameters settings structure `tBusParameter` see section *“Detailed Description of Bus and Master Parameters”.*

Note: This packet belongs to layer 7 (application layer) of the OSI-layer model for networks. There is also a packet with similar functionality, which allows setting bus parameters and is located at level 2 (the data link layer). The is packet described in section PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set of this document.

Note: The former parameter `bAlarm_Max` is no longer supported. It has been replaced by the new parameter `bMasterSetting`.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_BUS_PARAMETER_SET_Ttag {
  TLR_UINT16 usBus_Para_Len;
  TLR_UINT8  bDL_Add;
  TLR_UINT8  bData_rate;
  TLR_UINT16 usTSL;
  TLR_UINT16 usMin_TSDR;
  TLR_UINT16 usMax_TSDR;
  TLR_UINT8  bTQUI;
  TLR_UINT8  bTSET;
  TLR_UINT32 ulTTR;
  TLR_UINT8  bG;
  TLR_UINT8  bHSA;
  TLR_UINT8  bMax_Retry_Limit;
  TLR_UINT8  bBp_Flag;
  TLR_UINT16 usMin_Slave_Interval;
  TLR_UINT16 usPoll_Timeout;
  TLR_UINT16 usData_Control_Time;
  /*TLR_UINT8  bAlarm_Max; *//* Maximum Alarms */
  TLR_UINT8  bMasterSetting; /* 0 == big Endian, 1 == little Endian)*/
  TLR_UINT8  bMax_User_Global_Control;
  TLR_UINT8  abReserved[4];
  TLR_UINT16 usMaster_User_Data_Len;
  TLR_UINT8  abMaster_Class2_Name[32];
  TLR_UINT8  abMaster_User_Data[32];
  TLR_UINT32 ulTCL;
  TLR_UINT8  bMax_TSH;
} PROFIBUS_DL_BUS_PARAMETER_SET_T;

typedef struct PROFIBUS_FSPMM_INIT_REQ_Ttag
{
  PROFIBUS_DL_BUS_PARAMETER_SET_T tBusParameter;
}PROFIBUS_FSPMM_INIT_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_INIT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_INIT_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_INIT_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_INIT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle of AP-Task process queue
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle of FSPMM-Task process queue
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	n	sizeof(PROFIBUS_FSPMM_INIT_REQ_T)- Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.2 Error Codes of the FSPMM-Task*
ulCmd	UINT32	0x2200	`PROFIBUS_FSPMS_CMD_GLOBAL_CONTROL_IND` - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	×	Routing, do not change
structure `PROFIBUS_FSPMM_INIT_REQ_T`			
tBusParameter	STRUCT		Bus parameter settings

Table 47: PROFIBUS_FSPMM_CMD_INIT_REQ – Request for setting of Bus Parameters

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_PACKET_INIT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_FSPMM_PACKET_INIT_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_INIT_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.2 Error Codes of the FSPMM-Task*
ulCmd	UINT32	0x2201	PROFIBUS_FSPMM_CMD_INIT_CNF- Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 48: PROFIBUS_FSPMM_CMD_INIT_CNF - Confirmation Command for Request for Setting of Bus Parameter

6.1.3 PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode

The command changes the current operation mode of the PROFIBUS DP master stack. Possible operation modes are

- OFFLINE
- STOP
- CLEAR
- OPERATE

For a detailed explanation and discussion of these operation modes, see section *PROFIBUS DP Operation Modes* on page 14.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Packet Structure Reference

```
#define USIF_OFFLINE 0x00
#define USIF_STOP    0x40
#define USIF_CLEAR   0x80
#define USIF_OPERATE 0xC0

typedef struct PROFIBUS_FSPMM_SET_MODE_REQ_Ttag
{
  TLR_UINT32 ulBusMode;
}PROFIBUS_FSPMM_SET_MODE_REQ_T;

#define PROFIBUS_FSPMM_SET_MODE_REQ_SIZE sizeof(PROFIBUS_FSPMM_SET_MODE_REQ_T)

typedef struct PROFIBUS_FSPMM_PACKET_SET_MODE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_SET_MODE_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_SET_MODE_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_SET_MODE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	FSPMM_QUE	Destination queue handle, unchanged
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle, unchanged
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, unchanged
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.2 Error Codes of the FSPMM-Task*
ulCmd	UINT32	0x2206	PROFIBUS_FSPMM_CMD_SET_MODE_REQ_T - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_SET_MODE_REQ_T			
ulBusMode	UINT32	0,0x40,0x80, 0xC0	Mode that shall be set

Table 49: PROFIBUS_FSPMM_CMD_SET_MODE_REQ– Set a new operation mode

Packet Structure Reference

```
#define PROFIBUS_FSPMM_SET_MODE_CNF_SIZE sizeof(PROFIBUS_FSPMM_SET_MODE_CNF_T)

typedef struct PROFIBUS_FSPMM_SET_MODE_CNF_Ttag
{
  TLR_UINT32 ulBusMode;
}PROFIBUS_FSPMM_SET_MODE_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_SET_MODE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_SET_MODE_CNF_T tData;
} PROFIBUS_FSPMM_PACKET_SET_MODE_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_SET_MODE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.2 Error Codes of the FSPMM-Task*
ulCmd	UINT32	0x2207	PROFIBUS_FSPMM_CMD_MODE_CHANGE_CNF_T - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_MODE_CHANGE_CNF_T			
ulBusMode	UINT32		Active bus mode

Table 50: PROFIBUS_FSPMM_CMD_SET_MODE_CNF – Confirmation to Set a new operation mode

6.1.4 PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication

This indication is sent to the AP-task when the mode of the PROFIBUS DP Master has changed. The possible values of `ulBusMode` have the following meaning:

Value	State
0x00	USIF_OFFLINE
0x40	USIF_STOP
0x80	USIF_CLEAR
0xC0	USIF_OPERATE

For a detailed explanation and discussion of these operation modes, see section 3.2 *"PROFIBUS DP Operation Modes"* of this document.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_MODE_CHANGE_IND_Ttag
{
  TLR_UINT32 ulBusMode;
}PROFIBUS_FSPMM_MODE_CHANGE_IND_T;

typedef struct PROFIBUS_FSPMM_PACKET_MODE_CHANGE_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_MODE_CHANGE_IND_T tData;
}PROFIBUS_FSPMM_PACKET_MODE_CHANGE_IND_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_MODE_CHANGE_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.2 Error Codes of the FSPMM-Task*
ulCmd	UINT32	0x2214	`PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND` - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_MODE_CHANGE_IND_T			
ulBusMode	UINT32		Active bus mode

Table 51: PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication

6.1.5 PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic

This packet allows reading out the diagnostic structure of a slave. The desired address of the slave must be placed in `ulRemAdd` corresponding to the real address of the slave within the network.

The parameter `ulFlags` has influence to the kind of execution of this command. The flag `PROFIBUS_FSPMM_GET_DIAG_FLAG_QUERY` decides if either the diagnostic data is taken from the internal buffer of the master without causing a network access to the slave (set flag to 0), or if the service is sent to the slave directly (set flag to 1) and data of the response is sent back in the answer message. Calling the command without network access is much faster in reaction time and the preferred method for getting the answer than the other mode. However, using this mode makes only sense for slaves that are handled by the master. Only for those slaves the master drives the buffer update mechanism on every received new diagnostic data. The corresponding bit of each master assigned slave within the `Sl_Diag` field in the global bus status field indicates, if a changed diagnostic information since the last request is available and should be requested and read out. The diag bit of a slave station will be cleared on each `PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ` that is performed. `Sl_Diag`-bits of slaves, which are not assigned to the master, are not influenced by this mechanism in general.

The `Sl_Diag` field will be updated in relation to the executed main PROFIBUS master cycle time, which is needed to update all input and output data. If a slave station is permanently not available in the network for example, then the corresponding diag bit within this field could be set every 500 µsec repeatedly to indicate that the slave is not present.

The meaning of the single bits in the `ulFlags` parameter is:

Explanation of Bits of `ulFlags` Bit Mask			
Bit	**Name**	**Value**	**Description**
D31…D2	**Reserved**		Reserved, set to zero
D1	**QUERY**	0	Get slave diagnostic from masters internal buffer (Preferred method)
		1	Get diagnostic direct from slave (has influence to the bus cycle time)
D0	**PEEK**	0	Corresponding Diag Flag in `Sl_Diag` will be cleared by execution of the get diag command
		1	Corresponding Diag Flag in `Sl_Diag` will **not** be cleared by execution of the get diag command (useful diagnostic tools)

Table 52: ulFlags Bit Mask

Packet Structure Reference

```
#define PROFIBUS_FSPMM_GET_DIAG_FLAG_PEEK   0x00000001
#define PROFIBUS_FSPMM_GET_DIAG_FLAG_QUERY  0x00000002

typedef struct PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulFlags;
}PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_REQ_T;

#define PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_SIZE
(sizeof(PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8	Packet data length in bytes PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_SIZE
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x220A	PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_T			
ulRemAdd	UINT32	0 … 126	Requested slave address
ulFlag	UINT32	Bit mask	Flag register

Table 53: PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ – Request a Slave Diagnostic

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_DIAGNOSTIC_DATA_Ttag {
struct
{
TLR_UINT8 bStation_Non_Existent : 1; /* no response */
TLR_UINT8 bStation_Not_Ready : 1; /* station not ready */
TLR_UINT8 bCfg_Fault : 1; /* configuration fault */
TLR_UINT8 bExt_Diag : 1; /* extended diagnostic */
TLR_UINT8 bNot_Supported : 1; /* sync, freeze not supported */
TLR_UINT8 bInvalid_Response : 1; /* response faulty */
TLR_UINT8 bPrm_Fault : 1; /* parameters faulty */
TLR_UINT8 bMaster_Lock : 1; /* locked by a master */
} Stationstatus_1;
struct
{
TLR_UINT8 bPrm_Req : 1; /* request new parameters */
TLR_UINT8 bStat_Diag : 1; /* static diagnostic */
TLR_UINT8 bTrue : 1; /* set to 1 by a slave */
TLR_UINT8 bWd_On : 1; /* watchdog function on/off */
TLR_UINT8 bFreeze_Mode : 1; /* freeze mode active */
TLR_UINT8 bSync_Mode : 1; /* sync mode active */
TLR_UINT8 bReserved : 1; /* reserved */
TLR_UINT8 bDeactivated : 1; /* slave deactivated */
} Stationstatus_2;
struct
{
TLR_UINT8 bReserved : 7;
TLR_UINT8 bExt_Diag_Overflow : 1; /* ext. diagnostic overflow */
} Stationstatus_3;
TLR_UINT8 bMaster_Add; /* corresponding master address */
TLR_UINT16 usIdent_Number; /* ident number, Motorola format */
TLR_UINT8 abExt_Diag_Data[PROFIBUS_FSPM_ACT_MAX_EXT_DIAG_LEN];
/* extended diagnostic field */ } PROFIBUS_FSPMM_DIAGNOSTIC_DATA_T;

typedef struct PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  PROFIBUS_FSPMM_DIAGNostIC_DATA_T  tDiagnostic;
}PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_T;

#define PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_SIZE
(sizeof(PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_T) - sizeof(PROFIBUS_FSPMM_DIAGNOSTIC_DATA_T) )

typedef struct PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	4 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x220B	PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_T			
ulRemAdd	UINT32		Slave address
tDiagnostic	PROFIBUS_FSPMM_DIAGNostIC_DATA_T		Slave diagnostic

Table 54: PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_CNF – Confirmation of Request a Slave Diagnostic

The confirmation message informs the Host application about success or failure of the PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_CNF service.

Note: The internal buffer of the PROFIBUS DP device can handle 6 Bytes standard and 238 Bytes Extended Diagnostic data per slave device.

6.1.6 PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic

This indication signals that a slave has new diagnostic data available. `ulRemAdd` corresponds to the real address of the slave within the network.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_NEW_SLAVE_DIAG_IND_Ttag
{
  TLR_UINT32 ulRemAdd;
}PROFIBUS_FSPMM_NEW_SLAVE_DIAG_IND_T;

typedef struct PROFIBUS_FSPMM_PACKET_NEW_SLAVE_DIAG_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_NEW_SLAVE_DIAG_IND_T tData;
}PROFIBUS_FSPMM_PACKET_NEW_SLAVE_DIAG_IND_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_NEW_SLAVE_DIAG_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2216	PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_NEW_SLAVE_DIAG_IND_T			
ulRemAdd	UINT32	0…125	Slave address indicating the slave which is able to provide a new diagnostic

Table 55: PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic

6.1.7 PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output Data

This packet can be used to set new output data with the packet interface. If the application does not use the I/O area of the stack, this packet can also set the output data.

Cyclic data transfer is discussed in more detail in section *Cyclic Data Transfer* on page 15.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_SET_OUTPUT_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM_SET_OUTPUT_REQ_T;

#define PROFIBUS_FSPMM_SET_OUTPUT_REQ_SIZE (sizeof(PROFIBUS_FSPMM_SET_OUTPUT_REQ_T) -
PROFIBUS_FSPM_MAX_IO_DATA_LEN)

typedef struct PROFIBUS_FSPMM_PACKET_SET_OUTPUT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_SET_OUTPUT_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_SET_OUTPUT_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_SET_OUTPUT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4 + PROFIBUS_FSPM_MAX_IO_DATA_LEN	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x220C	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ_T - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM__SET_OUTPUT_REQ_T			
ulRemAdd	UINT32	0…125	Address of slave to receive the data
abData	UINT8[]	X	Data that shall be set

Table 56: PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ– Set new Output Data

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_PACKET_SET_OUTPUT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_FSPMM_PACKET_SET_OUTPUT_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_SET_OUTPUT_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x220D	PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF_T - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM__SET_OUTPUT_CNF_T			
ulRemAdd	UINT32	0…125	Address of slave

Table 57: PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF – Confirmation of Setting new Output Data

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_PROFIBUS_FSPMM_ INVALID_SLAVE_ADDRESS (0xC038001DL)	Invalid slave address.

Table 58: PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF – Packet Status/Error

6.1.8 PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input Data

The packet can be used to read cyclic input data of a slave. Cyclic data transfer is discussed in more detail in section *Cyclic Data Transfer* on page 15.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_GET_INTPUT_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
}PROFIBUS_FSPMM_GET_INPUT_REQ_T;

#define PROFIBUS_FSPMM_GET_INPUT_REQ_SIZE (sizeof(PROFIBUS_FSPMM_GET_INPUT_REQ_T))

typedef struct PROFIBUS_FSPMM_PACKET_GET_INPUT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_GET_INPUT_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_GET_INPUT_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_GET_INPUT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x220E	PROFIBUS_FSPMM_CMD_GET_INPUT_REQ_T - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	X	Routing, do not change
structure PROFIBUS_FSPMM_GET_INPUT_REQ_T			
ulRemAdd	UINT32	0…125	Slave Address from which to receive data

Table 59: PROFIBUS_FSPMM_CMD_GET_INPUT_REQ - Get new Input Data

Packet Structure Reference

```
#define MSK_INPUT_DATA_STATUS        (0x03000000)
#define FLG_INPUT_DATA_STATUS_CLR    (0x01000000)
#define FLG_INPUT_DATA_STATUS_NIL    (0x02000000)
#define FLG_INPUT_DATA_SYNC_ERR      (0x00000001)
#define FLG_INPUT_DATA_DUPADR_ERR    (0x00000002)

typedef struct PROFIBUS_FSPMM_GET_INTPUT_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlaveDiag;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM_GET_INPUT_CNF_T;

#define PROFIBUS_FSPMM_GET_INPUT_CNF_SIZE (sizeof(PROFIBUS_FSPMM_GET_INPUT_CNF_T) -
PROFIBUS_FSPM_MAX_IO_DATA_LEN )

typedef struct PROFIBUS_FSPMM_PACKET_GET_INPUT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_GET_INPUT_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_GET_INPUT_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_GET_INPUT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x220F	PROFIBUS_FSPMM_CMD_GET_INPUT_CNF_T - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_GET_INPUT_CNF_T			
ulRemAdd	UINT32	0…125	Slave address
ulSlaveDiag	UINT32		Input data diagnostic information Byte0 = Stationstatus_1 of standard DP Slave diag Byte1 = Stationstatus_2 of standard DP Slave diag Byte2 = Stationstatus_3 of standard DP Slave diag Byte3 = Data Status
abData	UINT8[]		Input data

Table 60: PROFIBUS_FSPMM_CMD_GET_INPUT_CNF - Confirmation of Get new Input Data

The variable ulSlaveDiag contains diagnostic information about the quality of the received input data and diagnostic information about the PROFIBUS slave itself. The first 3 bytes corresponding the standard diagnostic of a slave according the PROFIBUS specification.

The 4 Byte contains two flags to indicate information about the input data quality. Therefore two flags are spent.

```
#define FLG_INPUT_DATA_STATUS_CLR    (0x01000000)
#define FLG_INPUT_DATA_STATUS_NIL    (0x02000000)
```

If both flags are 0 it means the master is in data exchange with the slave and the input data are valid and fresh.

If the first bit FLG_INPUT_DATA_STATUS_CLR is set, the input data from the slave are not valid, the master has set the input data internally to 0. This happens, when the master has no yet established a connection to the slave.

If the second bit FLG_INPUT_DATA_STATUS_NIL is set, it means the data have the last value that has been received from the slave, but they are not fresh or up to date. This can happen for example when the slave is in data exchange and the slave indicates for some cycles static diagnosis and after that the slave continues the data exchange. In this case the FLG_INPUT_DATA_STATUS_NIL will be set, and the user can decide what to do with the data. Set the input data to “0” in its application or consume the received data for a hold the last state mechanism.

Flag FLG_INPUT_DATA_SYNC_ERR indicates a data synchronization error.

6.1.9 PROFIBUS_FSPMM_CMD_READ_REQ/CNF – DP V1 Class1 Read Request

This packet is used for a read request to a data block of a DP V1 slave. A data block is addressed using slot and index numbers.

For more information refer to section *Acyclic Data Transfer* on page 17.

Explanations of the parameters:

1. The parameter Rem_Add (`ulRemAdd`) contains the station address of the slave.
2. The parameter Slot_Number (`ulSlot`) is used in the destination device for addressing the desired data block slot (typically a module).
3. The parameter Index (`ulIndex`) is used in the destination device for addressing the desired data block.
4. The parameter Length (`ulLength`) is the number of bytes of the data block that has to be read. If the server data block length is less than requested, then the length in the response will be the actual length of the data block. If the server data block length is greater or equal than requested then the response contains the same length of data. The DP Slave may answer with an error response if the data access is not allowed.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_READ_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
}PROFIBUS_FSPMM_READ_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_READ_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_READ_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_READ_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_READ_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	16	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2210	PROFIBUS_FSPMM_CMD_READ_REQ_T - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_READ_REQ_T			
ulRemAdd	UINT32	0 ...125	Slave address
ulSlot	UINT32	0 ...254	Requested slot
ulIndex	UINT32	0 ...254	Requested index
ulLength	UINT32	1 ...240	Requested data length Using value 240 in the request means that the slave returns the real length of the data block read. **Note:** In PROFIBUS, the general maximum is 240 bytes. The maximum value depends on the used slave device and can be lower than 240 bytes.

Table 61: PROFIBUS_FSPMM_CMD_READ_REQ –V1 Class1 Read Request

Packet Structure Reference

(not applicable in case of error `TLR_E_PROFIBUS_FSPMM_MSAC1_NRS`)

```
typedef struct PROFIBUS_FSPMM_READ_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM_READ_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_READ_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_READ_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_READ_CNF_T;
```

Packet Structure Reference

(applicable only in case of error `TLR_E_PROFIBUS_FSPMM_MSAC1_NRS`)

```
typedef struct PROFIBUS_FSPMM_READ_CNF_NEG_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT8 bErrorDecode;
  TLR_UINT8 bErrorCode1;
  TLR_UINT8 bErrorCode2;
}PROFIBUS_FSPMM_READ_CNF_NEG_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_READ_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	12 + n	Packet data length in bytes If ulSta is 0: ulLen is 12 + n (n bytes read). If ulSta unequal 0: ulLen is 12 or 15. Length 15 indicates that bErrorDecode, bErrorCode1, and bErrorCode2 is provided.
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2211	PROFIBUS_FSPMM_CMD_READ_CNF_T - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_READ_CNF_T			
ulRemAdd	UINT32	0 ...125	Slave address
ulSlot	UINT32	0 ...254	Slot
ulIndex	UINT32	0 ...254	Index
In case of success (ulSta=0)			
abData	UINT8[]		Requested data
In case of error (ulSta=`TLR_E_PROFIBUS_FSPMM_MSAC1_NRS`)			
bErrorDecode	UINT8	128	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0…255	ErrorCode1, see section 3.3.2.2.of this document
bErrorCode2	UINT8	0…255	ErrorCode2, meaning depends on bErrorCode1

Table 62: PROFIBUS_FSPMM_CMD_READ_CNF –Confirmation of V1 Class1 Read Request

6.1.10 PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – DP V1 Class1 Write Request

This packet is used for a write request to a data block of a DP V1 slave. A data block is addressed using slot and index numbers.

Explanations of the parameters of the request packet:

1. The parameter Rem_Add (`ulRemAdd`) contains the station address of the slave responder.
2. The parameter Slot_Number (`ulSlot`) is used in the destination device for addressing the desired data block slot (typically a module).
3. The parameter Index (`ulIndex`) is used in the destination device for addressing the desired data block.
4. The field `abData` contains the data, which has to be written into the data block and has a length of n bytes (n = 1 … 240). The packet data length is 12 + n.

The parameter Length (`ulLength`) is only in the confirmation packet and is the number of bytes that has been written. If the destination data block length is less than the amount of data tob e writen then the reply will contain an error message. The PROFIBUS DP Slave may answer with an error response if the data access is not allowed.

Note: Read the manual of the used slave device to get the length information of the data block (slot and index) to be written and use this length within this service.

Note: The variables `bErrorDecode`, `bErrorCode1` and `bErrorCode2` of the confirmation packet are only present in case of error `TLR_E_PROFIBUS_FSPMM_MSAC1_NRS,` otherwise they are missing.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_WRITE_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM_WRITE_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_WRITE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_WRITE_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_WRITE_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_WRITE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	12 + n	Packet data length in bytes n is the number of bytes to be written. This number must match the data block length of the used slave. n = 1 … 240 **Note:** In PROFIBUS, the general maximum is 240 bytes. The maximum value depends on the used slave device and can be lower than 240 bytes.
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2212	PROFIBUS_FSPMM_CMD_WRITE_REQ_T - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_WRITE_REQ_T			
ulRemAdd	UINT32	0 ... 125	Slave address
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
abData	UINT8[]		Data to be written

Table 63: PROFIBUS_FSPMM_CMD_WRITE_REQ– V1 Class1 Write Request

Packet Structure Reference

(not applicable in case of error `TLR_E_PROFIBUS_FSPMM_MSAC1_NRS`)

```
typedef struct PROFIBUS_FSPMM_WRITE_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
}PROFIBUS_FSPMM_WRITE_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_WRITE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_WRITE_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_WRITE_CNF_T;
```

Packet Structure Reference

(applicable only in case of error `TLR_E_PROFIBUS_FSPMM_MSAC1_NRS`)

```
typedef struct PROFIBUS_FSPMM_WRITE_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8 bErrorDecode;
  TLR_UINT8 bErrorCode1;
  TLR_UINT8 bErrorCode2;
}PROFIBUS_FSPMM_WRITE_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_WRITE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_WRITE_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_WRITE_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_WRITE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32		Packet data length in bytes
		16	If ulSta is 0: ulLen is 16.
		19	If ulSta unequal 0: ulLen is 16 or 19. Length 19 indicates that bErrorDecode, bErrorCode1, and bErrorCode2 is provided, which is for example the case if ulSta is TLR_E_PROFIBUS_FSPMM_MSAC1_NRS.
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 “Error Codes of the FSPMM-Task”
ulCmd	UINT32	0x2213	PROFIBUS_FSPMM_CMD_WRITE_CNF_T - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_WRITE_CNF_T			
ulRemAdd	UINT32	0…125	Slave address
ulSlot	UINT32	0…254	Slot
ulIndex	UINT32	0…254	Index
ulLength	UINT32	1…240	Length of written bytes
		0	0 in case of error
The following three variables are only present in case of error `TLR_E_PROFIBUS_FSPMM_MSAC1_NRS`			
bErrorDecode	UINT8	128	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0…255	ErrorCode1, see section *DP V1 Error Processing* (page 18).
bErrorCode2	UINT8	0…255	ErrorCode2, meaning depends on bErrorCode1.

Table 64: PROFIBUS_FSPMM_CMD_WRITE_CNF – Confirmation of V1 Class1 Write Request

Packet Status/Error

Definition / Value	Description	Error source	Help
TLR_S_OK 0x00000000	Status ok		
TLR_E_PROFIBUS_FSPMM_ INVALID_SLAVE_ADDRESS 0xC038001DL	Invalid slave address.		
	Resource unavailable	Slave	Slave has no buffer space left for the requested service
	Requested function of master is not activated within the slave	Slave	Slave is not activated in its DPV1support
	No answer data, although the slave has to response with data	Slave	Slave has not sent back any amount of data
	No response of the station	Slave	Check network wiring, check bus address of slave or baud rate support
	Master not into the logical token ring	Network in general	Check master DP-Address or highest-station-Address of other masters. Examine bus wiring to bus short circuits.
	No plausible reaction of remote partner	Slave	Slave does not work conforming to the DPV1 norm
	Negative response received, access denied	Slave	Access denied to requested data. Check Error_Code_1 and Error_Code_2 in response message to get closer error information
	Device is about to stop the DPV1-communication or the DPV1 is not in OPEN state	Host, configuration	DPV1 communications must be configured to be activated by the Device
	Device has stopped the DPV1-communication automatically	Slave	A previously addressed slave has responded with parameters not conforming to the norm
TLR_E_PROFIBUS_FSP MM_REJ_PS 0xC0380019L	A previous service is still in process	Host	Wait for the outstanding answer first. Parallel services are not allowed
	The length indicator msg.data_cnt oversteps maximum configured size	Host	Reduce length of message or enlarge maximum buffer size in SyCon or in slave data set
	Wrong parameter in request	Host	Check msg.function or msg.device_adr parameter of requested message

Table 65: PROFIBUS_FSPMM_CMD_WRITE_CNF – Packet Status/Error

In case of error, the two `Error_Codes` represent further detailed error information.

See section “DP V1 Error Processing” of this document for more information.

6.1.11 PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification

When a slave indicates an alarm to the PROFIBUS DP Master, this alarm notification will be send to the application.

By means of this service, an alarm message is transferred from the DPV1-Slave to the DPV1-Master.

1. `ulRem_Adr` contains the station address, which indicates the alarm.
2. `ulSlot_Number` indicates the source of the alarm. Here the range extends from 0 to 254. The value 255 is declared as reserved in the DPV1 specification.
3. `ulAlarm_Type` identifies the alarm type:

Value	Alarm Type
0	Reserved
1	Diagnostic_Alarm
2	Process_Alarm
3	Pull_Alarm
4	Plug_Alarm
5	Status_Alarm
6	Update_Alarm
7 - 31	Reserved
32 - 126	manufacturer-specific
127	Reserved

Table 66: Alarm Types

4. `ulSpecifier` (= Alarm_Spec_Ack) gives additional alarm information, e.g. an alarm appears, disappears or no further differentiation is possible or if the alarm requires an additional user acknowledge. For further explanations, see Table 67, Table 68, and Table 69 on page 111.
5. The `abDiagnostic_User_Data[]` field will contain the received diagnostic information from the DPV1-Slave transparently. For the meaning, see manufacturer-specific documentation of the slave.

Alarm_Spec_Ack

D7	D6	D5	D4	D3	D2	D1	D0
Sequence Number					Add_Ack	Alarm_Specifier	

Table 67: Flags of `Alarm_Spec_Ack` parameter

The `Alarm_Specifier` contain the following information:

D1	D0	Error and slot status
0	0	No further differentiation
0	1	Error appears and slot disturbed
1	0	Error disappears and slot is okay
1	1	Error disappears and slot is still disturbed

Table 68: Alarm_Specifier Bits D1 and D0

The `Add_Ack` contains the following information:

D2	Additional acknowledge
0	No additional user acknowledge required.
1	This alarm requires a separate user acknowledge additionally to the usual response. This can be done for instance by means of a write service.

Table 69: Add_Ack Bit D2

The `Sequence Number` is a unique identification of an alarm and has value range 0 – 31.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_ALARM_NOTIFICATION_IND_Ttag
{
  TLR_UINT32 ulRem_Adr;
  TLR_UINT32 ulAlarm_Type;
  TLR_UINT32 ulSlot_Number;
  TLR_UINT32 ulSpecifier;
  TLR_UINT8  abDiagnostic_User_Data[FSPMM_MAX_ALARM_LEN];
}PROFIBUS_FSPMM_ALARM_NOTIFICATION_IND_T;

typedef struct PROFIBUS_FSPMM_PACKET_ALARM_NOTIFICATIONG_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_ALARM_NOTIFICATION_IND_T tData;
}PROFIBUS_FSPMM_PACKET_ALARM_NOTIFICATION_IND_T;

#define PROFIBUS_FSPMM_ALARM_NOTIFICATION_IND_SIZE
(sizeof(PROFIBUS_FSPMM_ALARM_NOTIFICATION_IND_T)-FSPMM_MAX_ALARM_LEN)
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_ALARM_NOTIFICATION_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	16 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section *Error Codes of the FSPMM-Task* on page 280.
ulCmd	UINT32	0x221A	PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_ALARM_NOTIFICATION_IND_T			
ulRem_Adr	UINT32	0 … 125	Slave address of the slave causing the alarm
ulAlarm_Type	UINT32	1 … 6 or 32 … 126	Type of the received alarm. See *Table 66: Alarm Types* above.
ulSlot_Number	UINT32	0 … 254	Slot number of module causing the alarm
ulSpecifier	UINT32	Bit mask	Alarm specifier. See explanation above in this section.
abDiagnostic_User_Data	UINT8[]		Alarm diagnostic user data

Table 70: PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification

6.1.12 PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge

This packet needs to be sent in order to acknowledge a previously received PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND indication. The device will then send a corresponding message to the slave device to inform it that the host application has received and acknowledged the alarm.

With this answer message, the host gets the confirmation for its previously sent PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ. If no error is indicated (ulSta=0), the master has successfully sent back the PROFIBUS message to the slave device. Then the positive confirmation packet is issued.

Otherwise, a negative confirmation packet is returned. If the slave answers with an error telegram the data is attached to the confirmation.

The following table of applicable error codes may be useful for troubleshooting purposes:

ulSta	Description	Error source	Help
TLR_E_PROFIBUS_FSPMM_MSALM_ALARM_ACK_NEG	The slave answers telegram with error telegram; Device is reconfigured	Device	Check error bytes of confirmation
TLR_E_PROFIBUS_FSPMM_REJ_ABORT	The slave answers with wrong telegram or timeout; Device is reconfigured	Device	Error on Device

Table 71: Possible Errors and Descriptions

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_ALARM_ACK_REQ_Ttag
{
  TLR_UINT32 ulRem_Adr;
  TLR_UINT32 ulAlarm_Type;
  TLR_UINT32 ulSlot_Number;
  TLR_UINT32 ulSpecifier;
  TLR_UINT32 ulFunction;
}PROFIBUS_FSPMM_ALARM_ACK_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_ALARM_ACK_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_ALARM_ACK_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_ALARM_ACK_REQ_T;

#define PROFIBUS_FSPMM_ALARM_ACK_REQ_SIZE (sizeof(PROFIBUS_FSPMM_ALARM_ACK_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_ALARM_ACK_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	20	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 “Error Codes of the FSPMM-Task”
ulCmd	UINT32	0x221C	PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ_T - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_ALARM_ACK_REQ_T			
ulRem_adr	UINT32	0 … 125	Slave address of the slave causing the alarm
ulAlarm_Type	UINT32	1 … 6 or 32 … 126	Type of the received alarm. See *Table 66: Alarm Types* above.
ulSlot_Number	UINT32	0 … 254	Slot number of module causing the alarm
ulSpecifier	UINT32	Bit mask	Alarm specifier. See explanation in previous section.
ulFunction	UINT32		Not used

Table 72: PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ – Alarm Acknowledge

Packet Structure Reference

```
/* Positive confirmation packet */

typedef struct PROFIBUS_FSPMM_ALARM_ACK_CNF_Ttag
{
  TLR_UINT32 ulRem_Adr;
  TLR_UINT32 ulAlarm_Type;
  TLR_UINT32 ulSlot_Number;
  TLR_UINT32 ulSpecifier;
  TLR_UINT32 ulFunction;
}PROFIBUS_FSPMM_ALARM_ACK_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_ALARM_ACK_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_ALARM_ACK_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_ALARM_ACK_CNF_T;

#define PROFIBUS_FSPMM_ALARM_ACK_CNF_SIZE (sizeof(PROFIBUS_FSPMM_ALARM_ACK_CNF_T))
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_ALARM_ACK_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	20	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32	0	
ulCmd	UINT32	0x221D	PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_ALARM_ACK_CNF_T			
ulRem_adr	UINT32	0 … 125	Slave address of the slave causing the alarm
ulAlarm_Type	UINT32	1 … 6 or 32 … 126	Type of the received alarm. See Table 66: Alarm Types above.
ulSlot_Number	UINT32	0 … 254	Slot number of module causing the alarm
ulSpecifier	UINT32	Bit mask	Alarm specifier. See explanation in previous section.
ulFunction	UINT32		Not used

Table 73: PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF – Confirmation of Alarm Acknowledge

Packet Structure Reference

```
/* Negative confirmation packet */

typedef struct PROFIBUS_FSPMM_ALARM_ACK_NEG_CNF_Ttag
{
  TLR_UINT32 ulRem_Adr;
  TLR_UINT32 ulAlarm_Type;
  TLR_UINT32 ulSlot_Number;
  TLR_UINT32 ulSpecifier;
  TLR_UINT32 ulFunction;
  TLR_UINT8 bErrorDecode;
  TLR_UINT8 bErrorCode1;
  TLR_UINT8 bErrorCode2;
  TLR_UINT8 bReserved;
}PROFIBUS_FSPMM_ALARM_ACK_NEG_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_ALARM_ACK_NEG_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_ALARM_ACK_NEG_CNF_T tData;
} PROFIBUS_FSPMM_PACKET_ALARM_ACK_NEG_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_ALARM_ACK_NEG_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	 20 24	Packet data length in bytes at ulSta = TLR_E_PROFIBUS_FSPMM_REJ_ABORT at ulSta = TLR_E_PROFIBUS_FSPMM_MSALM_ALARM_ACK_NEG
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32	!=0	See Table 71: Possible Errors and Descriptions
ulCmd	UINT32	0x221D	PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_ALARM_ACK_NEG_CNF_T			
ulRem_Adr	UINT32	0 … 125	Slave address of the slave causing the alarm
ulAlarm_Type	UINT32	1 … 6 or 32 … 126	Type of the received alarm. See *Table 66: Alarm Types* above.
ulSlot_Number	UINT32	0 … 254	Slot number of module causing the alarm
ulSpecifier	UINT32	Bit mask	Alarm specifier. See explanation in previous section.
ulFunction	UINT32	0	Not used
The following three variables are only present in case of `ulSta = TLR_E_PROFIBUS_FSPMM_MSALM_ALARM_ACK_NEG`			
bErrorDecode	UINT8	128	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0…255	ErrorCode1, see section 3.3.2.2.of this document
bErrorCode2	UINT8	0…255	ErrorCode2, meaning depends on bErrorCode1
bReserved	UINT8	0	Reserved

Table 74: PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF – Negative Confirmation of Alarm Acknowledge

6.1.13 PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ/CNF – Download Slave Parameter Set

This packet provides the possibility to download a new slave parameter set to a specific PROFIBUS DP slave.

The variable `ulAreaCode` is used to specify the address of the slave, which will receive the slave parameter block.

The slave parameter set is stored in variable `abData[]`. Structure and contents of slave parameter sets are extensively discussed in section 4.3 "Detailed Description of Slave Parameters" of this document.

The following DPV2 features can be accessed via the PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ command by downloading special parameter sets:

- Clock synchronization / Time stamp
- Slave to slave communication (DXB = Data Exchange Broadcast)

In both cases, special data have to be written into the `User_prm_data` field (i.e. bits 11 and following of parameter `Prm_Data` of the slave parameter set, see *Prm_Data* on page 65) which stores extended user specific data.

For clock synchronization, download for each slave which is planned to participate in synchronization a parameter set as described in section 4.3 with a `User_prm_data` field containing these data for the `User_prm_data` field:

Header

```
TLR_UINT8 Structure_Length
TLR_UINT8 Structure_Type
TLR_UINT8 Slot_Number
TLR_UINT8 reserved
```

Data

```
TLR_UINT16 Clock_Sync_Interval
TLR_UINT32 NetworkTimeDifferenceValueLo
TLR_UINT32 NetworkTimeDifferenceValueHi
```

Caution: Data have to be specified in Motorola format!

Use the following values within this header:

Structure_Length	14
Structure_Type	8
Slot_Number	0
Reserved	0

In the data part, specify the clock sync interval in units of 10 ms.

The network time difference means the difference of the current time to a reference time (fix point IEEE time). NetworkTimeDifferenceValueHi contains the seconds having passed since the reference time, NetworkTimeDifferenceValueLo the fractional part.

For DXB (Slave-to-slave cross communication), two cases must be considered separately:

1. Configuration for Publisher
2. Configuration for Subscriber

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_DOWNLOAD_REQ_Ttag
{
  TLR_UINT32 ulRemoteAddress;
  TLR_UINT32 ulAreaCode;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_PCKT_LEN];
}PROFIBUS_FSPMM_DOWNLOAD_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_DOWNLOAD_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_DOWNLOAD_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_DOWNLOAD_REQ_T;

#define PROFIBUS_FSPMM_DOWNLOAD_REQ_SIZE (sizeof(PROFIBUS_FSPMM_DOWNLOAD_REQ_T) -
PROFIBUS_FSPM_MAX_PCKT_LEN)
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_DOWNLOAD_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x221E	PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_DOWNLOAD_REQ_T			
ulRemoteAddress	UINT32	0xFF 0…125	Local master Address of other slave to communicate with (only DXB)
ulAreaCode	UINT32	0…125	Slave address
abData	UINT8[]		Slave parameter set, see structure below and detailed description in section 4.3.

Table 75: PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ– Download Slave Parameter Set

Structure of the Slave Parameter Set

Variable	Type	Value / Range	Description
abSlaveAdr	UINT8	0…125	SlaveAddress
usParmeterSetLength	UINT16	0…65535	Length of parameter set (i.e. the length of whole data set inclusive the length parameter)
bSlFlags	UINT8	0…255	Slave flags (in accordance to the DP extension to EN 50170(DPV1 support))
bSlaveType	UINT8	0…255	Slave type (should always be 0 for a DP…Slave)
bMaxDiagDataLen	UINT8		Maximum of diagnostic data length
bMaxAlarmLen	UINT8	4…64	Maximum length of a Alarm PDU
bMaxChannelDateLen	UINT8	4…244	Maximum size of channel date
bDiagUpdDelay	UINT8	0…15	Number of requests for a slave diagnostic (PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ) while request for parameters is still set
bAlarmMode	UINT8	0…7	Maximum number of possible active alarms
bAddSlFlag	UINT8	0…3	Ignore auto_clear
usC1Timeout	UINT16	0…65535	Timeout for C1 services
bReserved[4]	UINT8		Reserved for further use
usPrmDataLen	UINT16	9…246	Length of the following abPrmData area including the length of the size indicator
abPrmData	UINT8		see corresponding HEADER file for structure
usCfgDataLen	UINT16	3…246	Length of the following abCfgData area including the length of the size indicator
abCfgData	UINT8		see corresponding HEADER file for structure
usAddTabLen	UINT16	2…65504	Length of the following abAddTab data area including the length of the size indicator
abAddTab	UINT8		see corresponding HEADER file for structure
usSlaveUserDataLen	UINT16	2…65504	Length of the following slave user specific data area (abSlaveUserData) including the length of the size indicator
abSlaveUserData	UINT8		see corresponding HEADER file for structure

Table 76: Structure of the Slave Parameter Set according to IEC 61158 Specification

This structure corresponds to the structure described in the chapter 'coding of the slave parameters' of the PROFIBUS-norm IEC 61158.

- The `abPrmData` field includes the parameter data block, which will be sent to the slave station during its start-up procedure with the PROFIBUS DP command 'Set_Prm'. This field includes 7-byte norm specific parameters and a `User_prm_data` field as extended user specific data.
- The `abCfgData` field includes the configuration data of the slave, which decides on the number of input and outputs of the slave. This data is sent to the slave with the PROFIBUS DP command 'Check_Cfg' to induce the slave to compare this configuration with its own internally saved one.
- The `abAddTab` field is a Hilscher-specific field, which configures the different offset addresses of process data within the dual-port memory for modular and simple I/O slaves in common. See the following structure:

abAddTab Structure

Variable name	Type	Explanation
`Input_Count`	byte	number of input offsets following in the IO_Offset table
`Output_count`	byte	number of output offsets following in the IO_Offset table
`IO_Offsets[...]`	word array	word or byte IO_Offset in the order: first all input offsets then all output offsets in case of modular stations

Table 77: abAddTab Structure

One module entry in the `Cfg_list` must result in a corresponding entry in the `abAddTab`, except modules with a data length of 0. In this table, the offset address within the dual-port memory of each module is held down. If the upper bit 15 in the `IO_Offset[...]` value is set to logical '1' then the address is interpreted by the Device as byte offset address, else it is interpreted as word offset address.

Download example of a slave parameter data set with the address 4, without using the sequenced download procedure.

	command message			
	variable	**type**	**value**	**signification**
message header	ulDest	UINT32	0x20	Destination queue handle (receiver)
	ulSrc	UINT32	0	Source queue handle (transmitter)
	ulDestId	UINT32		Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source end point identifier, specifying the origin of the packet inside the source process
	ulLen	UINT32	???	Packet data length in bytes
	ulId	UINT32	j	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status
	ulCmd	UINT32	PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ	PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ - Command
	ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
	ulRout	UINT32		Routing, do not change
Service Header	ulRemoteAddress	UINT32	0xFF	Local master
	ulAreaCode	UINT32	4	Area code, denoting the slave address
	abData	UINT8[]	0	Slave parameter set
Slave parameter set	abSlaveAdr	UINT8		SlaveAddress
	usParmeterSetLength	UINT16		Length of parameter set
	bSlFlags	UINT8	128	Sl_Flag
	bSlaveType	UINT8	0	Slave_Type
	bMaxDiagDataLen	UINT8		Maximum of diagnostic data length
	bMaxAlarmLen	UINT8	4…64	Maximum length of a Alarm PDU
	bMaxChannelDateLen	UINT8	0	Maximum size of channel date
	bDiagUpdDelay	UINT8		Number of requests for a slave diagnostic (PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ) while request for parameters is still set
	bAlarmMode	UINT8	0…7	Maximum number of possible active alarms
	bAddSlFlag	UINT8		Ignore auto_clear
	usC1Timeout	UINT16		Timeout for C1 services
	bReserved[4]	UINT8		Reserved for further use
	usPrmDataLen	UINT16		Length of the following abPrmData area including the length of the size indicator
	abPrmData	UINT8		
	usCfgDataLen	UINT16		Length of the following abCfgData area including the length of the size indicator

	abCfgData	UINT8		
	usAddTabLen	UINT16		Length of the following abAddTab data area including the length of the size indicator
	abAddTab	UINT8		
	usSlaveUserDataLen	UINT16		Length of the following slave user specific data area (abSlaveUserData) including the length of the size indicator
	abSlaveUserData	UINT8		

Table 78: Example of a Slave Parameter Data Set

The coding at `bSlFlags` corresponds to the structure in the chapter 'coding of the slave parameters' in the PROFIBUS-norm IEC 61158.

Note: The structure of the slave parameters could not be laid down statically, because the data in the slave parameter set causes different lengths. Therefore no obvious addresses can be fixed to the several start addresses of the different parameters.

Packet Structure Reference

```
 typedef struct PROFIBUS_FSPMM_DOWNLOAD_CNF_Ttag
{
  TLR_UINT32 ulRemoteAddress;
  TLR_UINT32 ulAreaCode;
}PROFIBUS_FSPMM_DOWNLOAD_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_DOWNLOAD_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_DOWNLOAD_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_DOWNLOAD_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_DOWNLOAD_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x221F	PROFIBUS_FSPMM_CMD_DOWNLOAD_CNF- Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_FSPMM_DOWNLOAD_CNF_T			
ulRemoteAddress	UINT32	0xFF	Address of master
ulAreaCode	UINT32	0…125	Slave address

Table 79: PROFIBUS_FSPMM_CMD_DOWNLOAD_CNF – Configuration of Download Slave Parameter Set

6.1.14 PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF – Global Control Message

This packet can be used to send a global control message to one or several DP slaves. This is usually done periodically after one or more bus cycles.

The global control service provides the following possibilities:

- Inform the slave about the activity of the master
- Freeze or unfreeze the inputs of the slave.
- Synchronize or unsynchronize the outputs of the slave.

For more information on freezing or synchronizing inputs or outputs see sections 3.3.1.3 and 3.3.1.4 of this document.

A DP slave accepts a control command only from the DP master, which has parameterized the DP slave previously.

The parameter `ulRemAdd` determines the address of the slave to which the packet will be directed. Its value can range from 0 to 127.

- If a value between 0 and 126 is chosen, only one individual slave will be influenced.
- The value of 127 represents the special global address. If this address is selected, all slaves will be influenced simultaneously. This is also called a ‘broadcast’.

The parameters `ulSyncCommand` and `ulFreezeCommand` determine the command to be sent.

Global_Control_Command

Bit	Description
D0	Reserved: 0
D1	Clear_Data: clear output data
D2	UnFreeze: unfreeze input data
D3	Freeze: freeze input data
D4	Unsync: neutralize the sync command
D5	Sync: freeze output data, until sync command is neutralized
D6	0
D7	0

Table 80: Explanation of Bits in Global Control_Command

Explanations of possible combinations of the bits unsync/sync and unfreeze/freeze:

Combinations of unsync/sync and unfreeze/freeze bits

bit 2 or 4	bit 3 or 5	signification
0	0	no function
0	1	function will be activated
1	0	function will be inactive
1	1	function will be inactive

Table 81: Explanations of possible Combinations of the Bits 'Unsync'/'Sync' and 'Unfreeze'/'Freeze':

The parameter `Group_Select` decides which group of the possible eight ones is addressed. The command will be activated in the slave device, if the AND relation between its internal group identification parameter (the parameter that was configured by the master in the start-up phase of the slave) and the desired `Group_Select` logically represents the value '1'. If `Group_Select` contains the value ‘0’, no group selection (AND relation) is performed by the slave device when it receives the command.

GroupSelect

Bit								Meaning
7	6	5	4	3	2	1	0	
							1	Group 1
						1		Group 2
1								Group 8
1	1							Groups 7 and 8
0	0	0	0	0	0	0	0	All slaves

Table 82: GroupSelect

Bit D1 “`Clear_Data`” cannot be set by the user. Setting this flag has no effect to global control command on the network. The stack generates this bit by itself in dependency of its operate mode (CLEAR/OPRERATE). The user has no influence to this bit. The bit is mentioned here only for completeness of all global control flags.

Note: The following special aspect should be taken into account concerning the confirmation: The `PROFIBUS_FSPMM_GLOBALCONTROL_CNF` command will be sent as a multicast command. Therefore, this command always has a subsequent execution. This has the consequence that no error will be placed in the status variable `ulSta` of the answer message.

Also refer to the packet structure reference just below.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_GLOBALCONTROL_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulSyncCommand;
  TLR_UINT32 ulFreezeCommand;
  TLR_UINT32 ulGroupSelect;
}PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_GLOBALCONTROL_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_GLOBALCONTROL_REQ_T;

#define PROFIBUS_FSPMM_GLOBALCONTROL_REQ_SIZE sizeof(PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T)
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_GLOBALCONTROL_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	16	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2220	PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T			
ulRemAdd	UINT32	0—126 127	Slave address 127 represents the global address
ulSyncCommand	UINT32	0,16,32	Sync command 0x00: No Sync 0x10: Unsync 0x20: Sync
ulFreezeCommand	UINT32	0,4,8	Freeze command 0x00: No Freeze 0x04: Unfreeze 0x08: Freeze
ulGroupSelect	UINT32	0-255	Group selection

Table 83: PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ– Send a Global Control Request

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_PACKET_GLOBALCONTROL_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_FSPMM_PACKET_GLOBALCONTROL_CNF_T;

#define PROFIBUS_FSPMM_GLOBALCONTROL_CNF_SIZE 0
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_GLOBALCONTROL_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2221	PROFIBUS_FSPMM_CMD_GLOBALCONTROL_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 84: PROFIBUS_FSPMM_CMD_GLOBALCONTROL_CNF – Confirmation of Sending a Global Control Message

6.1.15 PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ/CNF – Activate/Deactivate Slaves

This packet can be used to activate or deactivate a slave during runtime. This means the master will start or stop the cyclic communication with a single slave on the network. The cyclic communication with all other slaves are continued by the master without stopping the hole network operation. This functionality can be used for maintenance work when a slave must be replaced in the network or to activate a backup slave in the network.

Packet Structure Reference

```
#define PROFIBUS_FSPMM_SLAVE_ACTIVATE    (0x00000001)
#define PROFIBUS_FSPMM_SLAVE_DEACTIVATE  (0x00000002)

typedef struct PROFIBUS_FSPMM_SLAVE_ACTIVATE_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulActivate;
}PROFIBUS_FSPMM_SLAVE_ACTIVATE_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_SLAVE_ACTIVATE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_SLAVE_ACTIVATE_REQ_T tData;
}PROFIBUS_FSPMM_PACKET_SLAVE_ACTIVATE_REQ_T;

#define PROFIBUS_FSPMM_SLAVE_ACTIVATE_REQ_SIZE
 sizeof(PROFIBUS_FSPMM_SLAVE_ACTIVATE_REQ_T)
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_SLAVE_ACTIVATE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2226	PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T			
ulRemAdd	UINT32	0—125	Slave address
ulActivate	UINT32	1 2	Activate Slave Deactivate Slave

Table 85: PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ – Slave Activate Request Packet

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_SLAVE_ACTIVATE_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulActivate;
}PROFIBUS_FSPMM_SLAVE_ACTIVATE_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_SLAVE_ACTIVATE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_SLAVE_ACTIVATE_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_SLAVE_ACTIVATE_CNF_T;

#define PROFIBUS_FSPMM_SLAVE_ACTIVATE_CNF_SIZE
 sizeof(PROFIBUS_FSPMM_SLAVE_ACTIVATE_CNF_T)
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_SLAVE_ACTIVATE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2227	PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_CNF - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_SLAVE_ACTIVATE_CNF_T			
ulRemAdd	UINT32	0—125	Slave address
ulActivate	UINT32	1 2	Activate Slave Deactivate Slave

Table 86: PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_CNF – Slave Activate Confirmation Packet

6.1.16 PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a Fatal Fault

This packet indicates an unrecoverable fault at the master behavior. When it occurs, the state machine is turned back to the “Power on” state (`USIF_OFFLINE`).

Possible reasons for the fault are:

- DDLM_Fault
- Set_Slave_Add
- MSAC1S_Fault

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_FAULT_IND_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulFault;
}PROFIBUS_FSPMM_FAULT_IND_T;

typedef struct PROFIBUS_FSPMM_PACKET_FAULT_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_FAULT_IND_T tData;
}PROFIBUS_FSPMM_PACKET_FAULT_IND_T;

#define PROFIBUS_FSPMM_FAULT_IND_SIZE (sizeof(PROFIBUS_FSPMM_FAULT_IND_T))
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_FAULT_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter *Packets* for Configuration during running system.
ulCmd	UINT32	0x2222	PROFIBUS_FSPMM_CMD_FAULT_IND - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM_FAULT_IND_T			
ulRemAdd	UINT32	0 … 127	Slave address where the fault occurred
ulFault	UINT32		Number of fault

Table 87: PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a fatal Fault

6.1.17 PROFIBUS_MSCS1M_CMD_INIT_CS_REQ/CNF – Initialize ClockSync Feature

The *init* command has to be sent once before using the *set time* command. It is used to register the callbacks needed for the *ClockSync* feature.

Packet Structure Reference

```
typedef struct PROFIBUS_MSCS1M_PACKET_INIT_CS_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_MSCS1M_PACKET_INIT_CS_REQ_T;
```

Packet Description

structure PROFIBUS_MSCS1M_PACKET_INIT_CS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x222C	PROFIBUS_MSCS1M_CMD_INIT_CS_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

Table 88: PROFIBUS_MSCS1M_CMD_INIT_CS_REQ – Initialize ClockSync Feature

Packet Structure Reference

```
typedef struct PROFIBUS_MSCS1M_PACKET_INIT_CS_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_MSCS1M_PACKET_INIT_CS_CNF_T;
```

Packet Description

structure PROFIBUS_MSCS1M_PACKET_INIT_CS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPMS0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x222D	PROFIBUS_MSCS1M_CMD_INIT_CS_CNF- Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

Table 89: PROFIBUS_MSCS1M_CMD_INIT_CS_CNF – Confirmation to Initialize ClockSync Feature Request

6.1.18 PROFIBUS_MSCS1M_CMD_SET_TIME_REQ – Set Time for *TimeEvent ClockValue* Sequence

The command *set time* is used to send the *TimeEvent ClockValue* sequence.

- The parameter ulTimeValueSec refers to the count of seconds passed since 1.1.1900 or, if the value is less than 0x9DFF4400, since 7.2.2036 6:28:16 (DD.MM.YYYY)
- The parameter ulTimeValueNsec equals the additional nanoseconds.
- The parameter lTimeDiffMin must be a multiple of 30 (minutes) in the range from -930 to 930.
- The parameter fSummerTime is true if the timezone is in summer time, false if in winter time.
- The parameter ulAccuracyMSec must be one of the following:
  1, 10, 1000, 1000 (milliseconds).
- The parameter fSyncActive is true if the ClockSync is synchronized with other clocks, false if not synchronized.
- The parameter fAnnouncementHour is true if a change of winter/summer time will occur within the next hour.

Further Information can be found in IEC-61158-6-3.

The parameters `ulSystimeRefSec` and `ulSystimeRefNSec` should be used only, if access to the netX systime is possible, to link the time with the netX systime. By default the linking is made in APM Task.

This packet requires initialization by once sending the PROFIBUS_MSCS1M_CMD_INIT_CS_REQ/CNF – Initialize ClockSync Feature packet, see description on page 132.

Packet Structure Reference

```
typedef struct PROFIBUS_MSCS1M_SET_TIME_REQ_Ttag
{
  TLR_UINT32  ulTimeValueSec;
  TLR_UINT32  ulTimeValueNSec;
  TLR_INT32   lTimeDiffMin;
  TLR_BOOLEAN fSummerTime;
  TLR_UINT32  ulAccuracyMSec;
  TLR_BOOLEAN fSyncActive;
  TLR_BOOLEAN fAnnouncementHour;
  TLR_UINT32  ulSystimeRefSec;
/*Only if Apm Task is not used (rcX time reference)*/
  TLR_UINT32  ulSystimeRefNSec;
/*Only if Apm Task is not used (rcX time reference)*/
}PROFIBUS_MSCS1M_SET_TIME_REQ_T;

typedef struct PROFIBUS_MSCS1M_PACKET_SET_TIME_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_MSCS1M_SET_TIME_REQ_T tData;
}PROFIBUS_MSCS1M_PACKET_SET_TIME_REQ_T;
```

Packet Description

structure PROFIBUS_MSCS1M_PACKET_SET_TIME_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulFSPMM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	28 or 36	Packet data length in bytes, depends on access to the netX systime: 19 without netX systime 27 with netX systime
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x222E	PROFIBUS_MSCS1M_CMD_SET_TIME_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_MSCS1M_SET_TIME_REQ_T;			
ulTimeValueSec	UINT32	0 ... 2^{32}-1	Time value (in seconds, see explanation above)
ulTimeValueNSec	UINT32	0 ... 2^{32}-1	Time value fraction (in nanoseconds, see explanation above)
lTimeDiffMin	INT32	-930,-900,…, -30,0,30,…,930	Time difference (see explanation above)
fSummerTime	BOOLEAN	FALSE, TRUE	Timezone (i.e. summer time or winter time)
ulAccuracyMSec	UINT32	1, 10, 1000, 1000	Accuracy (in milliseconds)
fSyncActive	BOOLEAN	FALSE, TRUE	Sync active
fAnnouncementHour	BOOLEAN	FALSE, TRUE	Announcement Hour (see explanation above)
ulSystimeRefSec	UINT32	0 ... 2^{32}-1	System Time Reference (in seconds)
ulSystimeRefNSec	UINT32	0 ... 2^{32}-1	System Time Reference fraction (in nanoseconds)

Table 90: PROFIBUS_MSCS1M_CMD_SET_TIME_REQ – Set Time Request

Packet Structure Reference

```
typedef struct PROFIBUS_MSCS1M_PACKET_SET_TIME_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_MSCS1M_PACKET_SET_TIME_CNF_T;
```

Packet Description

structure PROFIBUS_MSCS1M_PACKET_SET_TIME_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x222F	PROFIBUS_MSCS1M_CMD_SET_TIME_CNF - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

Table 91: PROFIBUS_MSCS1M_CMD_SET_TIME_CNF – Confirmation to Set Time Request

6.1.19 PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_REQ/CNF – Switch Redundancy

To use the feature Slave Redundancy (System Redundancy), the `PrmCmd` structure with the `CheckProperties` flag set has to be included in the `PrmData` of the SlaveParameter data set.

To (de)activate a Slave or a set of Slaves the command redundancy switch has to be sent.

The packet contains the section of the `PrmCmd` structure to be sent to the slaves.

Additionally, a bit list with a size of 128 entries permits sending the PrmCmd to multiple slaves.

To address a slave, the PROFIBUS address is used as offset. (Slave 1 is at position `abAddress[0]` bit 1, slave 31 is at position `abAddress [3]` bit 7)

The confirmation packet contains the bit list from the request, a bit list with executed PrmCmd and a bit list with Errors on PrmCmd (Slave not configured).

The slaves responds with a diagnosis sequence at switchover or other possible PrmCmds, the evaluation has to be done by the application.

Packet Structure Reference

```
typedef struct PROFIBUS_PRMCMD_Ttag
{
  TLR_UINT8 bLen;
  TLR_UINT8 bType;
  TLR_UINT8 bSlotNo;
  TLR_UINT8 bSpecifier;
  TLR_UINT8 bFunction;
  TLR_UINT8 bProperties;
  TLR_UINT8 bOutputHoldTimeHighByte;
  TLR_UINT8 bOutputHoldTimeLowByte;
} PROFIBUS_PRMCMD_T;

typedef struct PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_REQ_Ttag
{
  PROFIBUS_PRMCMD_T tPrmCmd;
  TLR_UINT8  abAddress[16];
} PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_REQ_T;

typedef struct PROFIBUS_MCSY1M_PACKET_REDUNDANCY_SWITCH_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_REQ_T tData;
}PROFIBUS_MSCY1M_PACKET_REDUNDANCY_SWITCH_REQ_T;
```

Packet Description

structure PROFIBUS_MSCY1M_PACKET_REDUNDANCY_SWITCH_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	ulFSPMM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	24	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2230	PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_REQ_T			
tPrmCmd	PROFIBUS_PRMCMD_T		PrmCmd structure
abAddress[16]	UINT8[]		Address structure

Table 92: PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_REQ – Switch Redundancy

Packet Structure Reference

```
typedef struct PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_CNF_Ttag
{
  TLR_UINT8  abAddress[16];
  TLR_UINT8  abExecute[16];
  TLR_UINT8  abError[16];
} PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_CNF_T;

typedef struct PROFIBUS_MSCY1M_PACKET_REDUNDANCY_SWITCH_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_CNF_T tData;
} PROFIBUS_MSCY1M_PACKET_REDUNDANCY_SWITCH_CNF_T;
```

Packet Description

structure PROFIBUS_MSCY1M_PACKET_REDUNDANCY_SWITCH_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	48	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2231	PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_MSCY1M_REDUNDANCY_SWITCH_CNF_T			
abAddress[16]	UINT8[]		Address structure
abExecute[16]	UINT8[]		Execute structure
abError[16]	UINT8[]		Error structure

Table 93: PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_CNF – Confirmation to Switch Redundancy Packet

6.1.20 PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_REQ/CNF - Input Change of Slaves

The feature *Input Change of Slaves* is a bit list, after the bit lists of configured/data exchange/diagnosis of slaves after the input image in DPM, which contains the slaves, where the inputs changed since last DPM update. Slaves who change their inputs back and forth during one DPM update cycle are registered, too.

It is possible to enable the feature per slave in order to reduce the system load.

To enable/disable the feature the command UPDATE_ICOS_CONFIG is used.

The bit list in the packet is used to address the slaves by the PROFIBUS address. (Slave 8 is at position `abAddress[1]` bit 0, slave 125 is at position `abAddress[15]` bit 5). A one enables the feature, a zero disables it.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_ICOS_REQ_Ttag
{
  TLR_BOOLEAN fIrqSuppression;
  TLR_UINT8 abAddress[16];
}PROFIBUS_FSPMM_ICOS_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_ICOS_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_ICOS_REQ_T tData;
} PROFIBUS_FSPMM_PACKET_ICOS_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_ICOS_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	ulFSPMM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	16	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2232	PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_FSPMM_ICOS_REQ_T			
fIrqSuppression	BOOLEAN	0,1	Decides whether IRQ Suppression is used or not
abAddress[16]	UINT8[]		Address structure

Table 94: PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_REQ - Input Change of Slaves Request

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_PACKET_ICOS_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_FSPMM_PACKET_ICOS_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_ICOS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2233	PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 95: PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_CNF – Confirmation to Input Change of Slaves Request

6.1.21 PROFIBUS_FSPMM_CMD_UPLOAD_REQ/CNF – Upload Slave Parameter Set

This packet provides the possibility to upload a slave parameter set from a specific PROFIBUS DP slave or the bus parameter set.

The value of `ulRemoteAddress` must be always 0xFF.

The variable `ulAreaCode` is used to specify the address of the slave, which parameters will be uploaded.

The variable `ulOffset` is used to read the data from an offset value with length specified by `ulLength` variable.

Structure and contents of slave parameter sets are extensively discussed in section *Detailed Description of Slave Parameters* on page 62 of this document.

Note: The first item of the Slave Parameter Set (i.e. the Slave address) as described there is missing here. The Slave Parameter Set directly begins with the second item (Length of parameter set).

If `ulAreaCode` equals 127, the master parameter set is selected, see `PROFIBUS_DL_BUS_PARAMETER_SET_T` in section 6.2.1.

Packet Structure Reference

```
/* Upload slave parameter set*/
typedef struct PROFIBUS_FSPMM_UPLOAD_REQ_Ttag
{
  TLR_UINT32 ulRemoteAddress;
  TLR_UINT32 ulAreaCode;
  TLR_UINT32 ulOffset;
  TLR_UINT32 ulLength;
} PROFIBUS_FSPMM_UPLOAD_REQ_T;

typedef struct PROFIBUS_FSPMM_PACKET_UPLOAD_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_UPLOAD_REQ_T tData;
} PROFIBUS_FSPMM_PACKET_UPLOAD_REQ_T;

#define PROFIBUS_FSPMM_UPLOAD_REQ_SIZE (sizeof(PROFIBUS_FSPMM_UPLOAD_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_UPLOAD_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	ulFSPMM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32		Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2234	PROFIBUS_FSPMM_CMD_UPLOAD_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_FSPMM_UPLOAD_REQ_T			
ulRemoteAddress	UINT32	0xFF	Remote Address
ulAreaCode	UINT32	0…125, 127	Area Code 127: Bus parameter set is transferred instead of slave parameter set
ulOffset	UINT32		Offset of Slave Parameter Set
ulLength	UINT32		Length of Slave Parameter Set

Table 96: PROFIBUS_FSPMM_CMD_UPLOAD_REQ – Upload Slave Parameter Set Request

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM_UPLOAD_CNF_Ttag
{
  TLR_UINT32 ulRemoteAddress;
  TLR_UINT32 ulAreaCode;
  TLR_UINT32 ulOffset;
  TLR_UINT32 ulLength;
  TLR_UINT8 abData[PROFIBUS_FSPM_MAX_PCKT_LEN];
}PROFIBUS_FSPMM_UPLOAD_CNF_T;

typedef struct PROFIBUS_FSPMM_PACKET_UPLOAD_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM_UPLOAD_CNF_T tData;
}PROFIBUS_FSPMM_PACKET_UPLOAD_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM_PACKET_UPLOAD_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32		Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.2 Error Codes of the FSPMM-Task
ulCmd	UINT32	0x2235	PROFIBUS_FSPMM_CMD_UPLOAD_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_FSPMM_UPLOAD_CNF_T			
ulRemoteAddress	UINT32	0xFF	Remote Address
ulAreaCode	UINT32	0…125, 127	Area Code
ulOffset	UINT32		Offset of Slave Parameter Set
ulLength	UINT32		Length of Slave Parameter Set
abData[PROFIBUS_FSPM_MAX_PCKT_LEN]	UINT8[]		Array containing the requested data

Table 97: PROFIBUS_FSPMM_CMD_UPLOAD_CNF – Confirmation of Upload Slave Parameter Set Request

6.2 The FSPMM2-Task

Within the DP-Master Stack, the FSPMM2-Task coordinates the underlying DP-Master state machines for DP V1 Class2 Masters used for processing of the various services. This allows to maintain communication relationships between the Hilscher device acting as a PROFIBUS DP V1 Class2 master and a slave.

Furthermore, it is responsible for all application interactions and represents the counterpart of the AP-Task within the existent DP-Master Stack implementation.

To get the handle of the process queue of the FSPMM2-Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII queue name	Description
"FSPMM2_QUE"	Name of the FSPMM2-Task process queue

Table 98: FSPMM-Task process queue

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the FSPMM2-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the FSPMM-Task.

In detail, the following functionality is provided by the FSPMM-Task:

Overview over Packets of the FSPMM2 -Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.2.1	PROFIBUS_FSPMM2_CMD_INIT_REQ/CNF – Initialization Command	0x4400/ 0x4401	147
6.2.2	PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate DPV1 Class2 Connection	0x4404/ 0x4405	149
6.2.3	PROFIBUS_FSPMM2_CMD_READ_REQ/CNF – DP V1 Class2 Read Request	0x4406/ 0x4407	157
6.2.4	PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class2 Write Request	0x4408/ 0x4409	162
6.2.5	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request	0x440A/ 0x440B	167
6.2.6	PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection	0x440C/ 0x440D	172
6.2.7	PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ/CNF – Read Slave Diagnostics (DP V1 Class2)	0x440E/ 0x440F	176
6.2.8	PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ/CNF – Read Input Values	0x4410/ 0x4411	178
6.2.9	PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ/CNF – Read Output Values	0x4412/ 0x4413	180
6.2.10	PROFIBUS_FSPMM2_CMD_GET_CFG_REQ/CNF – Get Configuration Command	0x4414/ 0x4415	182
6.2.11	PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ/CNF – Set Slave Address (DP V1 Class2)	0x4416/ 0x4417	184
6.2.12	PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ/CNF – Get Master Diagnosis	0x4418/ 0x4419	187
6.2.13	PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ/CNF – Live List Command	0x4430/ 0x4431	189
6.2.14	PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication	0x4426/ 0x4427	192
6.2.15	PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication	0x4428/ 0x4429	195
6.2.16	PROFIBUS_FSPMM2_CMD_RESET_CONN_REQ/CNF – Reset Connection Command	0x4432/ 0x4433	197

Table 99: Overview over the Packets of the FSPMM-Master -Task of the PROFIBUS DP-Master Protocol Stack

6.2.1 PROFIBUS_FSPMM2_CMD_INIT_REQ/CNF – Initialization Command

This service needs to be send from the AP-Task in order to initialize the stack. For this purpose, the bus parameter set needs to be send prior with PROFIBUS_FSPMM_CMD_INIT_REQ see section 6.1.2

The corresponding confirmation packet does not have any parameters.

Note: Sending bus parameters with PROFIBUS_FSPMM2_CMD_INIT_REQ results in an error TLR_E_PROFIBUS_FSPMM2_NOT_IMPLEMENTED and does not initialize the FSPMM2 state machine.

Packet Description

structure PROFIBUS_FSPMM2_PACKET_INIT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4400	PROFIBUS_FSPMM2_CMD_INIT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 100: PROFIBUS_FSPMM2_CMD_INIT_REQ- Initialization Command for Class2 Connection

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM2_PACKET_INIT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_FSPMM2_PACKET_INIT_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_INIT_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4401	PROFIBUS_FSPMM2_CMD_INIT_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change

Table 101: PROFIBUS_FSPMM2_CMD_INIT_CNF - Confirmation of Initialization Command

6.2.2 PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate DPV1 Class2 Connection

The FSPMM2-Task is managing one instance of the “MSAC2M State machine” for each acyclic DPV1 Class2 connection to a slave. This state machine is responsible for performing and supervising the acyclic data transfer between the DP-Class2 Master and a DP-Class2 Slave. The FSPMM2-Task has to be requested first with the PROFIBUS_FSPMM2_CMD_INITIATE_REQ” command to establish a connection before any following acyclic communication can be performed to a DP-Class2 Slave.

- PROFIBUS_FSPMM2_CMD_READ_REQ/CNF – DP V1 Class2 Read Request
- PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class2 Write Request
- PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request

The parameter ulRem_Add is used for storing the station address in order to provide access protection.

The time-out usSend_Timeout specifies the control time interval for the supervision of the DPV1 Class2 connection (MSAC_C2 connection) which the DP V1 Master Class2 requests from the slave. The supervision based on this period is performed as long as the DPV1 Class2 connection is active. The time out values are specified in units of 10 milliseconds.

According to the PROFIBUS specification, PROFIBUS DP V1 Master Class2 and PROFIBUS DP V1 Slave inform each other about their supported service functionality using the variables bFeaturesSupported1 and bFeaturesSupported2. This process gives both the slave and the AP-Task the opportunity to adjust its functionality to the current requirements or to reject the request if it cannot fulfill them.

This process works as follows:

The 'Initiate' service sends the parameter directly to the slave which immediately checks this parameter. If a slave does not support a requested service or a defined profile, it will deny the 'Initiate' request. The connection state will change into the 'Closed' state then. If the initiate request is confirmed positively then the connection state go to 'Opened' and the master will now request Idle telegrams to the slave cyclically to supervise it, until the connection is aborted or interrupted physically. In the last case the master will automatically abort and close the connection. In a Class2 relationship only one open connection can exist at the same time, connections established in parallel are forbidden. The device will deny further 'Initiate' requests as long as a Class2 connection to a slave is in 'Opened' state.

To initialize a connection to another slave device this already established connection must be aborted before with the PROFIBUS_FSPMM2_CMD_ABORT_REQ request. A connection is in 'Opened' state as long as the device has not send an PROFIBUS_FSPMM2_CMD_CLOSED_IND indication message to the host. That means between this PROFIBUS_FSPMM2_CMD_INITIATE_REQ request message and the PROFIBUS_FSPMM2_CMD_CLOSED_IND indication message all messages like the PROFIBUS_FSPMM2_CMD_ABORT_IND indication or PROFIBUS_FSPMM2_CMD_XXX_CNF_NEG messages have to be collected by the host program too. Only when the PROFIBUS_FSPMM2_CMD_CLOSED_IND messages was received, no further messages must be awaited by the host for this previously established Class2 connection.

The bits of the bFeaturesSupported1 variable have the following meaning:

- Bit 0: Must be set in order to indicate support for services MSAC2_Read and MSAC2_Write which is mandatory for all DP V1 slaves.
- Bit 1-7: Reserved

All bits of the bFeaturesSupported2 variable are reserved, too.

The bit fields bProfileFeaturesSupported1 and bProfileFeaturesSupported2 inform the AP-Task about the requested service functionality regarding the used profile definition (if a profile is used, i.e. .usProfileIdentNumber is not equal to 0). The profile is identified by the Profile Ident number. The meaning of the defined bits is profile or vendor specific.

The variable usProfileIdentNumber identifies a profile definition uniquely. All devices using the same profile definition have to use the same profile ident number. The profile ident number. will be taken from the pool of ident numbers for vendor specific or authorized profiles. The value 0 indicates the no profile is supported. If the requested profile is supported by the AP-Task, the profile ident number. is mirrored in the response. If the requested profile is not supported by the AP-Task, the AP-Task has to respond negatively or with the profile ident number. the AP-Task supports.

The additional address parameter consists of two parts containing information about the source and the destination.

For each of source and destination the following information is stored there:

- Additional address information
- Presence or absence of network/ address MAC address
- Length of the additional address information

The additional address information contains the following information:

- The application process instance (API) of the source or destination, respectively.
- Access level of the source/destination.
- Network address (optional)
- MAC address (optional)

For these values the following rules apply:

- The application process instance (API) of the source/destination is characterized by an 8-bit number. The range of possible values extends from 0 to 255
- The access level of the source/destination is also given by an 8-bit number in the range from 0 to 255.
- The optional values network address and MAC address of the source or destination, respectively, are only present, when the source type or destination type have the value 1.
- The network address must be a valid 6-byte network address according to ISO/OSI-rules.
- The MAC address is a string according to the rules for MAC addresses.

Depending on whether the connection could successfully be opened or not, the confirmation packet will have a nearly completely different data part. However, the header part will be the same. Successful execution is signified by a value of `ulSta` (within the head of the confirmation packet) equal to 0, while all non-zero values at this location indicate an error during connection establishment.

- In case of successful execution the description under `PROFIBUS_FSPMM2_INITIATE_CNF_POS_T` will apply. This indicates that the MSAC_C2 connection is now in the 'opened' state. A number will be assigned to the connection, namely the communication reference, which is used as local identifier for the state machine of this communication relationship. The communication reference is returned in variable `ulCRef` of `PROFIBUS_FSPMM2_INITIATE_CNF_POS_T`. This identifier is very important for subsequent read, write or data transport operations, so you should store it after receiving the confirmation packet. The maximum length of data transmissions the slave permits is delivered in variable `bMaxLenDataUnit` of the positive confirmation packet. The variables `bFeaturesSupported1`, `bFeaturesSupported2`, `bProfileFeaturesSupported1`, `bProfileFeaturesSupported2`, `usProfileIdentNumber` and `tAddAddrParam` will contain the same values as in the request packet if the slave agrees with the request features and profile specifications. Otherwise the deviations will indicate where specific features are not supported. This information originates from the slave itself.
- In case of failure the confirmation packet will be structured as described under `PROFIBUS_FSPMM2_INITIATE_CNF_NEG_T`. Variable `usDetail` will have a non-zero value in case of the send timeout of the packet being less than the timeout of the resource manager of the slave. The returned value is equal to the send timeout according to the specification (IEC61158-6-3, p. 322, see line #8). For an explanation of the error handling see section *DP V1 Error Processing* on page 18 and the packet description.

For more information also see section 3.3.5.2"*Diagnostic model of PROFIBUS DP V1*" on page 27.

Packet Structure Reference

```
#define PROFIBUS_FSPMM2_ADD_ADDR_TABLE_LEN 228
typedef struct PROFIBUS_FSPMM2_ADD_ADDR_PARAMtag
{
  TLR_UINT8  bS_Type;
  TLR_UINT8  bS_Len;
  TLR_UINT8  bD_Type;
  TLR_UINT8  bD_Len;
  TLR_UINT8  abAddParam[PROFIBUS_FSPMM2_ADD_ADDR_TABLE_LEN];
} PROFIBUS_FSPMM2_ADD_ADDR_PARAM;

typedef struct  PROFIBUS_FSPMM2_INITIATE_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT16 usSendTimeout;
  TLR_UINT8  bFeaturesSupported1;
  TLR_UINT8  bFeaturesSupported2;
  TLR_UINT8  bProfileFeaturesSupported1;
  TLR_UINT8  bProfileFeaturesSupported2;
  TLR_UINT16 usProfileIdentNumber;
  PROFIBUS_FSPMM2_ADD_ADDR_PARAM tAddAddrParam;
} PROFIBUS_FSPMM2_INITIATE_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_INITIATE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_INITIATE_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_INITIATE_REQ_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_INITIATE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	12+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4404	PROFIBUS_FSPMM2_CMD_INITIATE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_INITIATE_REQ_T			
ulRemAdd	UINT32	0...126	Remote Address
usSendTimeout	UINT16	1…65535	Send timeout value for periodic supervision of connection to be established
bFeaturesSupported 1	UINT8	0…255	Bit field containing information about the service functionality
bFeaturesSupported 2	UINT8	0…255	Bit field containing additional information about the service functionality
bProfileFeaturesSupported1	UINT8	0…255	Bit field containing information about the profile
bProfileFeaturesSupported2	UINT8	0…255	Bit field containing additional information about the profile features
usProfileIdentNumber	UINT16	0…65535	Identification number for profile (A value of 0 indicates no profile is currently used.)
tAddAddrParam	PROFIBUS_FSPMM2_ADD_ADDR_PARAM		Additional Address Parameter Table

Table 102: PROFIBUS_FSPMM2_CMD_INITIATE_REQ - Initiate Command

Packet Structure Reference

```
typedef union PROFIBUS_FSPMM2_INITIATE_CNF_Ttag
{
  PROFIBUS_FSPMM2_INITIATE_CNF_POS_T tCnfPos;
  PROFIBUS_FSPMM2_INITIATE_CNF_NEG_T tCnfNeg;
} PROFIBUS_FSPMM2_INITIATE_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_INITIATE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_INITIATE_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_INITIATE_CNF_T;
```

Packet Description

Either positive or negative confirmation applies, see there.

Packet Structure Reference

```
#define PROFIBUS_FSPMM2_ADD_ADDR_TABLE_LEN 228
typedef struct PROFIBUS_FSPMM2_ADD_ADDR_PARAMtag
{
  TLR_UINT8  bS_Type;
  TLR_UINT8  bS_Len;
  TLR_UINT8  bD_Type;
  TLR_UINT8  bD_Len;
  TLR_UINT8  abAddParam[PROFIBUS_FSPMM2_ADD_ADDR_TABLE_LEN];
} PROFIBUS_FSPMM2_ADD_ADDR_PARAM;

typedef struct  PROFIBUS_FSPMM2_INITIATE_CNF_POS_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT32 ulCRef;
  TLR_UINT8  bMaxLenDataUnit;
  TLR_UINT8  bFeaturesSupported1;
  TLR_UINT8  bFeaturesSupported2;
  TLR_UINT8  bProfileFeaturesSupported1;
  TLR_UINT8  bProfileFeaturesSupported2;
  TLR_UINT16 usProfileIdentNumber;
  PROFIBUS_FSPMM2_ADD_ADDR_PARAM tAddAddrParam;
} PROFIBUS_FSPMM2_INITIATE_CNF_POS_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_INITIATE_CNF_T			Type: Positive Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	15+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4405	PROFIBUS_FSPMM2_CMD_INITIATE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_INITIATE_CNF_POS_T			
ulRemAdd	UINT32	0...126	Remote Address
ulCRef	UINT32	0…65535	Communication Reference for uniquely identifying the connection
bMaxLenDataUnit	UINT8	0…255	Maximum length of data unit to transfer
bFeaturesSupported1	UINT8	0…255	Bit field containing information about the service functionality actually supported by the DP V1 slave
bFeaturesSupported2	UINT8	0…255	Bit field containing additional information about the service functionality actually supported by the DP V1 slave
bProfileFeaturesSupported1	UINT8	0…255	Bit field containing information about the profile actually supported by the DP V1 slave
bProfileFeaturesSupported2	UINT8	0…255	Bit field containing additional information about the profile features actually supported by the DP V1 slave
usProfileIdentNumber	UINT16	0…65535	Identification number for the actually chosen profile
tAddAddrParam	PROFIBUS_FSPMM2_ADD_ADDR_PARAM		Additional Address Parameter Table

Table 103: PROFIBUS_FSPMM2_CMD_INITIATE_CNF_POS - Positive Confirmation of Initiate Command

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_INITIATE_CNF_NEG_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  bErrorDecode;
  TLR_UINT8  bErrorCode1;
  TLR_UINT8  bErrorCode2;
  TLR_UINT8  bReserved;
  TLR_UINT16 usDetail;
  TLR_UINT16 usReserved;
} PROFIBUS_FSPMM2_INITIATE_CNF_NEG_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_INITIATE_CNF_T			**Type: Negative Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	12	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*. Typical error codes are: One DPV1 Class 2 connection is supported and a subsequent initiate request is rejected with error code TLR_E_PROFIBUS_FSPMM2_IN_USE, if the master is aware about another connection to slave or TLR_E_PROFIBUS_DL_ACK_RS, if the slave does not support class 2 services or all connections in the slave for DPV1 class 2 are used.
ulCmd	UINT32	0x4405	PROFIBUS_FSPMM2_CMD_INITIATE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_INITIATE_CNF_NEG_T			
ulRemAdd	UINT32	0...126	Remote Address
bErrorDecode	UINT8	128	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0…255	ErrorCode1, see section 3.3.2.2.
bErrorCode2	UINT8	0…255	ErrorCode2, meaning depends on bErrorCode1
bReserved	UINT8	0…255	Reserved
usDetail	UINT16	0…65535	Additional detailed error information.
usReserved	UINT16	0…65535	Reserved

Table 104: PROFIBUS_FSPMM2_CMD_INITIATE_CNF_NEG - Negative Confirmation of Initiate Command

6.2.3 PROFIBUS_FSPMM2_CMD_READ_REQ/CNF – DP V1 Class2 Read Request

This packet transfers a read request for a data block from the device acting as a PROFIBUS DP V1 Master Class2 to a DP V1-Slave. The applied addressing mechanism operates based on slot and index information.

Explanations of the parameters:

1. The parameter `ulCRef` contains the communication reference of the connection to the responding slave. This number has been returned at initialization by the confirmation packet `PROFIBUS_FSPMM2_CMD_INITIATE_CNF_POS`.
2. The parameter `ulSlot` = Slot_Number is used in the destination device for addressing the desired data block slot (typically a module).
3. The parameter `ulIndex` = `Index` is used in the destination device for addressing the desired data block.
4. The parameter `ulLength` = `Length` indicates the number of bytes of the data block that has to be read. If the server data block length is less than requested, then the length in the response will be the actual length of the data block. If the server data block length is greater or equal than requested then the response contains the same length of data. The DP-slave may answer with an error response if the data access is not allowed.

Depending on whether the connection could successfully be opened or not, the confirmation packet will have a nearly completely different data part. However, the header part will be the same. Successful execution is signified by a value of `ulSta` (within the head of the confirmation packet) equal to 0, while all non-zero values at this location indicate an error during connection establishment.

- In case of successful execution the description under `PROFIBUS_FSPMM2_READ_CNF_POS_T` will apply. This indicates that the read access to the slave over the `MSAC_C2` connection has been performed successfully.
- In case of failure the confirmation packet will be structured as described under `PROFIBUS_FSPMM2_ READ_CNF_NEG_T`. For an explanation of the error handling see section *DP V1 Error Processing* on page 18 and the packet description.

The positive confirmation packet contains the same parameters plus additionally the following one:

1. The field `abData` contains the data block, which has been read and consists of the number of octets indicated in the length parameter `ulLength` of the request.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_REQ_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
} PROFIBUS_FSPMM2_READ_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_READ_REQ_T;

#define PROFIBUS_FSPMM2_READ_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_READ_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	16	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4406	PROFIBUS_FSPMM2_CMD_READ_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_REQ_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	1 ... 240	Number of bytes of the data block that has to be read.

Table 105: PROFIBUS_FSPMM2_CMD_READ_REQ – Read Command

Packet Structure Reference

```
typedef union PROFIBUS_FSPMM2_READ_CNF_Ttag
{
  PROFIBUS_FSPMM2_READ_CNF_POS_T tCnfPos;
  PROFIBUS_FSPMM2_READ_CNF_NEG_T tCnfNeg;
} PROFIBUS_FSPMM2_READ_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_CNF_Ttag
{
  TLR_PACKET_HEADER_T        tHead;
  PROFIBUS_FSPMM2_READ_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_READ_CNF_T;
```

Packet Description

Either positive or negative confirmation applies, see there.

Packet Status/Error

This depends on positive or negative confirmation, see there.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_CNF_POS_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
} PROFIBUS_FSPMM2_READ_CNF_POS_T;

#define PROFIBUS_FSPMM2_READ_CNF_POS_SIZE (sizeof(PROFIBUS_FSPMM2_READ_CNF_POS_T)-
PROFIBUS_FSPM_MAX_IO_DATA_LEN)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_CNF_T			**Type: Positive Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	16+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4407	PROFIBUS_FSPMM2_CMD_READ_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_CNF_POS_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Number of bytes of the data block that has been read.
abData	UINT8[]		Data area containing the data that have been read. The length of this area corresponds to the value specified in variable `ulLength`.

Table 106: PROFIBUS_FSPMM2_CMD_READ_CNF_POS - Positive Confirmation of Read Command

Packet Status/Error

Definition / (Value)	**Description**
TLR_S_OK (0x00000000)	Status ok

Table 107: PROFIBUS_FSPMM2_CMD_READ_CNF_POS - Packet Status/ErrorPacket Description

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_CNF_NEG_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  bErrorDecode;
  TLR_UINT8  bErrorCode1;
  TLR_UINT8  bErrorCode2;
  TLR_UINT8  bReserved;
} PROFIBUS_FSPMM2_READ_CNF_NEG_T;

#define PROFIBUS_FSPMM2_READ_CNF_NEG_SIZE (sizeof(PROFIBUS_FSPMM2_READ_CNF_NEG_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_CNF_T			**Type: Negative Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	20	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4407	PROFIBUS_FSPMM2_CMD_READ_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_CNF_NEG_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Number of bytes of the data block that has been read.
bErrorDecode	UINT8	0 ... 255	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0 ... 255	ErrorCode1, see section 3.3.2.2.
bErrorCode2	UINT8	0 ... 255	ErrorCode2, meaning depends on bErrorCode1
bReserved	UINT8		

Table 108: PROFIBUS_FSPMM2_CMD_READ_CNF_NEG - Negative Confirmation of Read Command

6.2.4 PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class2 Write Request

This packet can be used to send a DP V1 Class2 write request to a PROFIBUS DP V1 slave. The applied addressing mechanism operates based on slot and index information.

Explanations of the parameters:

1. The parameter `ulCRef` contains the reference of the slave responder. This number has been returned at initialization by `PROFIBUS_FSPMM2_CMD_INITIATE_CNF`.
2. The parameter `ulSlot = Slot_Number` is used in the destination device for addressing the desired data block slot (typically a module).
3. The parameter `ulIndex = Index` is used in the destination device for addressing the desired data block.
4. The parameter `ulLength = Length` indicates the number of bytes of the data block that has to be written. If the server data block length is less than requested, then the length in the response will be the actual length of the data block. If the server data block length is greater or equal than requested then the response contains the same length of data. The DP-slave may answer with an error response if the data access is not allowed.
5. The field `abData` contains the data block, which has to be written and consists of the number of octets indicated in the length of the request.

Depending on whether the connection could successfully be opened or not, the confirmation packet will have a nearly completely different data part. However, the header part will be the same. Successful execution is signified by a value of `ulSta` (within the head of the confirmation packet) equal to 0, while all non-zero values at this location indicate an error during connection establishment.

- In case of successful execution the description under `PROFIBUS_FSPMM2_WRITE_CNF_POS_T` will apply. This indicates that the read access to the slave over the `MSAC_C2` connection has been performed successfully. The positive confirmation packet contains the same parameters as the request packet except the last one that has been omitted.
- In case of failure the confirmation packet will be structured as described under `PROFIBUS_FSPMM2_WRITE_CNF_NEG_T`. For an explanation of the error handling see section *DP V1 Error Processing* on page 18 and the packet description.

Packet Structure Reference

```
** WRITE REQ/CNF PACKET
*/
typedef struct  PROFIBUS_FSPMM2_WRITE_REQ_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
} PROFIBUS_FSPMM2_WRITE_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_WRITE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_WRITE_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_WRITE_REQ_T;

#define PROFIBUS_FSPMM2_WRITE_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_WRITE_REQ_T)-
PROFIBUS_FSPM_MAX_IO_DATA_LEN)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_WRITE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	16+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4408	PROFIBUS_FSPMM2_CMD_WRITE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_WRITE_REQ_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	1 ... 240	Number of bytes of the data block that has to be written.
abData	UINT8[]		Data area containing the data that should be written to the slave. The length of this area corresponds to the value specified in variable `ulLength`.

Table 109: PROFIBUS_FSPMM2_CMD_WRITE_REQ – Write Command

Packet Structure Reference

```
typedef union PROFIBUS_FSPMM2_WRITE_CNF_Ttag
{
  PROFIBUS_FSPMM2_WRITE_CNF_POS_T tCnfPos;
  PROFIBUS_FSPMM2_WRITE_CNF_NEG_T tCnfNeg;
} PROFIBUS_FSPMM2_WRITE_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_WRITE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_WRITE_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_WRITE_CNF_T;
```

Packet Description

Either positive or negative confirmation applies, see there.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_WRITE_CNF_POS_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
} PROFIBUS_FSPMM2_WRITE_CNF_POS_T;

#define PROFIBUS_FSPMM2_WRITE_CNF_POS_SIZE (sizeof(PROFIBUS_FSPMM2_WRITE_CNF_POS_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_WRITE_CNF_T			**Type: Positive Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	16	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4409	PROFIBUS_FSPMM2_CMD_WRITE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_WRITE_CNF_POS_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Number of bytes of the data block that has been written.

Table 110: PROFIBUS_FSPMM2_CMD_WRITE_CNF_POS - Positive Confirmation of Write Command

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_WRITE_CNF_NEG_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  bErrorDecode;
  TLR_UINT8  bErrorCode1;
  TLR_UINT8  bErrorCode2;
  TLR_UINT8  bReserved;
} PROFIBUS_FSPMM2_WRITE_CNF_NEG_T;

#define PROFIBUS_FSPMM2_WRITE_CNF_NEG_SIZE (sizeof(PROFIBUS_FSPMM2_WRITE_CNF_NEG_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_WRITE_CNF_T			**Type: Negative Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	20	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4409	PROFIBUS_FSPMM2_CMD_WRITE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_WRITE_CNF_NEG_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Requested length of data area
bErrorDecode	UINT8	0 ... 255	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0 ... 255	ErrorCode1, see section 3.3.2.2.
bErrorCode2	UINT8	0 ... 255	ErrorCode2, meaning depends on value of bErrorCode1
bReserved	UINT8		Reserved

Table 111: PROFIBUS_FSPMM2_CMD_WRITE_CNF_NEG - Negative Confirmation of Write Command

6.2.5 PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined DP V1 Class2 Read and Write Request

This packet can be used to send a data transport request, i.e. combined DP V1 Class2 read and write request.

Explanations of the parameters:

1. The parameter `ulCRef` contains the reference of the slave responder. This number has been returned at initialization by `PROFIBUS_FSPMM2_CMD_INITIATE_CNF`.
2. The parameter `ulSlot` = Slot_Number is used in the destination device for addressing the desired data block slot (typically a module).
3. The parameter `ulIndex` = `Index` is used in the destination device for addressing the desired data block.
4. The parameter `ulLength` = `Length` indicates the number of bytes of the data block that has to be read. If the server data block length is less than requested, then the length in the response will be the actual length of the data block. If the server data block length is greater or equal than requested then the response contains the same length of data. The DP-slave may answer with an error response if the data access is not allowed.
5. The field `abData` contains the data block, which has been read and consists of the number of octets indicated in the length parameter `ulLength` of the request.

Depending on whether the connection could successfully be opened or not, the confirmation packet will have a nearly completely different data part. However, the header part will be the same. Successful execution is signified by a value of `ulSta` (within the head of the confirmation packet) equal to 0, while all non-zero values at this location indicate an error during connection establishment.

- In case of successful execution the description under `PROFIBUS_FSPMM2_WRITE_CNF_POS_T` will apply. This indicates that the read access to the slave over the `MSAC_C2` connection has been performed successfully. The positive confirmation packet has exactly the same parameters with the same meanings.
- In case of failure the confirmation packet will be structured as described under `PROFIBUS_FSPMM2_WRITE_CNF_NEG_T`. For an explanation of the error handling see section *DP V1 Error Processing* on page 18 and the packet description.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_DATA_TRANSPORT_REQ_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM2_DATA_TRANSPORT_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_DATA_TRANSPORT_REQ_T tData;
}PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_REQ_T;

#define PROFIBUS_FSPMM2_DATA_TRANSPORT_REQ_SIZE
(sizeof(PROFIBUS_FSPMM2_DATA_TRANSPORT_REQ_T)-PROFIBUS_FSPM_MAX_IO_DATA_LEN)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	16+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x440A	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_DATA_TRANSPORT_REQ_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Number of bytes of the data block that has to be written.
abData	UINT8[]		The data block that has to be written itself.

Table 112: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ – Data Transport Command

Packet Structure Reference

```
typedef union PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_Ttag
{
  PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_POS_T tCnfPos;
  PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_T tCnfNeg;
} PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_CNF_T;

#define PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_SIZE
(sizeof(PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_T))
```

Packet Description

Either positive or negative confirmation applies, see there.

Packet Structure Reference

```
typedef struct PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_POS_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_POS_T;

#define PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_POS_SIZE
(sizeof(PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_POS_T)-PROFIBUS_FSPM_MAX_IO_DATA_LEN)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_CNF_T			**Type: Positive Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle, unchanged
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	16+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x440B	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_POS_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Number of bytes of the data block that has been read.
abData	UINT8[]		

Table 113: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF_POS - Positive Confirmation of Data Transport Command

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT32 ulSlot;
  TLR_UINT32 ulIndex;
  TLR_UINT32 ulLength;
  TLR_UINT8  bErrorDecode;
  TLR_UINT8  bErrorCode1;
  TLR_UINT8  bErrorCode2;
  TLR_UINT8  bReserved;
}PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_T;

#define PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_SIZE
(sizeof(PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_DATA_TRANSPORT_CNF_T			**Type: Negative Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	20	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task.* Parallel DPV1 transport request are not supported by master stack. If a request is in progress then any further request is rejected with error code TLR_E_PROFIBUS_FSPMM2_NOT_IN_OPEN_STATE. This error is also reported, if the connection is not open.
ulCmd	UINT32	0x440B	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_DATA_TRANSPORT_CNF_NEG_T			
ulCRef	UINT32		Connection Reference
ulSlot	UINT32	0 ... 254	Slot
ulIndex	UINT32	0 ... 254	Index
ulLength	UINT32	0 ... 240	Number of bytes of the data block that should have been read.
bErrorDecode	UINT8	0 ... 255	A value of 128 here indicates DP V1 error handling is applied.
bErrorCode1	UINT8	0 ... 255	ErrorCode1, see section 3.3.2.2.
bErrorCode2	UINT8	0 ... 255	ErrorCode2, meaning depends on bErrorCode1
bReserved	UINT8		

Table 114: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF_NEG - Negative Confirmation of Data Transport Command

6.2.6 PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection

Using this packet an abort of the DP V1 Class2 connection can be requested from the slave. If the corresponding confirmation is received, this confirms that the slave has received the request. If the connection has been aborted successfully, a separate PROFIBUS_FSPMM2_CMD_CLOSED_IND indication will be sent. This is a clear signal that the connection actually has been aborted.

The single parameters of the request packet have the following meaning:

- ulCRef
  This is the communication reference identifying the connection which has been delivered by the positive confirmation packet (PROFIBUS_FSPMM2_INITIATE_CNF_POS) from the ‘initiate’ command.
- bSubnet
  This variable indicates the source of the abort, i.e.. whether it originates from the local or a remote subnet or no specific source information should be given. The coding is as follows:

Subnet Variable

Value	Meaning
0	NO
1	SUBNET-LOCAL
2	SUBNET-REMOTE
3-255	Reserved

Table 115: Allowed Values of Subnet Variable

- bInstance
  The protocol instance causing the abort is specified here. Possible values are FDL, USER, MSAC_C2. The coding is as follows:

Instance Codes and their Meaning

D7	D6	D5	D4	Protocol Instance causing Abort
0	0	0	0	FDL
0	0	0	1	MSAC_C2
0	0	1	0	USER

Table 116: Instance Codes and their Meaning

- bReasonCode
  The reason code is a 4 bit value (between 0 and 15) whose meaning depends from the instance and indicates the reason for the abort according to the table below:

Available Reason Codes

D7	D6	D5	D4	D3 - D0	Instance	Reason Code	Description
Always 0	Always 0	0	0	1	FDL	UE	User Error
				2		RR	No FDL resource
				3		RS	Reject FDL service
				9		NR	No FDL response data
				10		DH	Data Reply High
				11		LR	see EN 50170 Part 2
				12		RDL	see EN 50170 Part 2
				13		RDH	see EN 50170 Part 2
				14		DS	Master is not in the logical ring
				15		NA	No response from remote FDL
		0	1	1	DDLM	ABT_SE	Sequence Error
				2		ABT_FE	Invalid request PDU received
				3		ABT_TO	Timeout of the connection
				4		ABT_RE	Invalid response PDU received
				5		ABT_IV	Invalid service received from USER
				6		ABT_STO	Send_Timeout requested was not large enough
				7		ABT_IA	Additional address information is not valid
				8		ABT_OC	Waiting for confirmation of FDL_DATA_REPLY
		1	0		User		User-caused abort

Table 117: Possible Reason Codes and their Meaning

- abReserved
- This is a reserved area.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_ABORT_REQ_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT8  bSubnet;
  TLR_UINT8  bInstance;
  TLR_UINT8  bReasonCode;
  TLR_UINT8  abReserved[1];
} PROFIBUS_FSPMM2_ABORT_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_ABORT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_ABORT_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_ABORT_REQ_T;

#define PROFIBUS_FSPMM2_ABORT_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_ABORT_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_ABORT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x440C	PROFIBUS_FSPMM2_CMD_ABORT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_ABORT_REQ_T			
ulCRef	UINT32	See above	Connection Reference
bSubnet	UINT8	0…2	Subnet
bInstance	UINT8	0…15	Instance number (4 bits)
bReasonCode	UINT8	0…15	Reason code why an abort is requested
abReserved[1]	UINT8		Reserved field

Table 118: PROFIBUS_FSPMM2_CMD_ABORT_REQ – Abort Request

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_ABORT_CNF_Ttag
{
  TLR_UINT32  ulCRef;
} PROFIBUS_FSPMM2_ABORT_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_ABORT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_ABORT_CNF_T tData;
}PROFIBUS_FSPMM2_PACKET_ABORT_CNF_T;

#define PROFIBUS_FSPMM2_ABORT_CNF_SIZE (sizeof(PROFIBUS_FSPMM2_ABORT_CNF_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_ABORT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x440D	PROFIBUS_FSPMM2_CMD_ABORT_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_ABORT_CNF_T			
ulCRef	UINT32		Connection Reference

Table 119: PROFIBUS_FSPMM2_CMD_ABORT_CNF - Confirmation of Abort Request

6.2.7 PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ/CNF – Read Slave Diagnostics (DP V1 Class2)

This packet requests a slave diagnostic from the PROFIBUS DP slave with the remote address specified in parameter `ulRemAdd`.

The diagnostic data are transferred into the field `tDiagnostic` of the confirmation packet and can be evaluated according to the rules explained above in section 3.3.5.2 "*Diagnostic model of PROFIBUS DP V1*" of this document.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_SLAVE_DIAG_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
}PROFIBUS_FSPMM2_READ_SLAVE_DIAG_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_SLAVE_DIAG_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_SLAVE_DIAG_REQ_T tData;
}PROFIBUS_FSPMM2_PACKET_READ_SLAVE_DIAG_REQ_T;


#define PROFIBUS_FSPMM2_READ_SLAVE_DIAG_REQ_SIZE
(sizeof(PROFIBUS_FSPMM2_READ_SLAVE_DIAG_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_SLAVE_DIAG_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x440E	PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_SLAVE_DIAG_REQ_T			
ulRemAdd	UINT32	0 …126	Remote address of PROFIBUS DP slave station from which diagnostic data shall be requested.

Table 120: PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ - Read Slave Diagnostics (DP V1 Class2)

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_SLAVE_DIAG_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  PROFIBUS_FSPMM_DIAGNOSTIC_DATA_T  tDiagnostic;
}PROFIBUS_FSPMM2_READ_SLAVE_DIAG_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_SLAVE_DIAG_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_SLAVE_DIAG_CNF_T tData;
}PROFIBUS_FSPMM2_PACKET_READ_SLAVE_DIAG_CNF_T;

#define PROFIBUS_FSPMM2_READ_SLAVE_DIAG_CNF_SIZE
(sizeof(PROFIBUS_FSPMM2_READ_SLAVE_DIAG_CNF_T) - sizeof(PROFIBUS_FSPMM_DIAGNOSTIC_DATA_T)
)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_SLAVE_DIAG_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	4+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x440F	PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_SLAVE_DIAG_CNF_T			
ulRemAdd	UINT32	0 …126	Remote address of PROFIBUS DP slave station from which diagnostic data shall be requested.
tDiagnostic	PROFIBUS_FSPMM_DIAGNOSTIC_DATA_T		Diagnostic data block

Table 121: PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_CNF - Confirmation of Read Slave Diagnostics (DP V1 Class2)

6.2.8 PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ/CNF – Read Input Values

This packet provides the Class2 master feature to read a PROFIBUS DP V1 slave's input values.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_INPUT_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
}PROFIBUS_FSPMM2_READ_INPUT_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_INPUT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_INPUT_REQ_T tData;
}PROFIBUS_FSPMM2_PACKET_READ_INPUT_REQ_T;

#define PROFIBUS_FSPMM2_READ_INPUT_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_READ_INPUT_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_INPUT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4410	PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_INPUT_REQ_T			
ulRemAdd	UINT32	0...126	Remote Address of the slave to read input data from

Table 122: PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ – Read Input Request

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_INPUT_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM2_READ_INPUT_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_INPUT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_INPUT_CNF_T tData;
}PROFIBUS_FSPMM2_PACKET_READ_INPUT_CNF_T;

#define PROFIBUS_FSPMM2_READ_INPUT_CNF_SIZE (sizeof(PROFIBUS_FSPMM2_READ_INPUT_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_INPUT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	4+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4411	PROFIBUS_FSPMM2_CMD_READ_INPUT_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_INPUT_CNF_T			
ulRemAdd	UINT32		Remote address of PROFIBUS DP slave station from which input data have been requested.
abData	UINT8[]		Area for input data that have been read

Table 123: PROFIBUS_FSPMM2_CMD_READ_INPUT_CNF – Confirmation of Read Input

6.2.9 PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ/CNF – Read Output Values

This packet provides the Class2 master feature to read a slave's output values.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_OUTPUT_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
}PROFIBUS_FSPMM2_READ_OUTPUT_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_OUTPUT_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_OUTPUT_REQ_T tData;
}PROFIBUS_FSPMM2_PACKET_READ_OUTPUT_REQ_T;

#define PROFIBUS_FSPMM2_READ_OUTPUT_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_READ_OUTPUT_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_OUTPUT_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4412	PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_OUTPUT_REQ_T			
ulRemAdd	UINT32	0...126	Remote address of PROFIBUS DP slave station from which output data shall be requested.

Table 124: PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ – Read Output

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_READ_OUTPUT_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
}PROFIBUS_FSPMM2_READ_OUTPUT_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_READ_OUTPUT_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_READ_OUTPUT_CNF_T tData;
}PROFIBUS_FSPMM2_PACKET_READ_OUTPUT_CNF_T;

#define PROFIBUS_FSPMM2_READ_OUTPUT_CNF_SIZE (sizeof(PROFIBUS_FSPMM2_READ_OUTPUT_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_READ_OUTPUT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	4+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4413	PROFIBUS_FSPMM2_CMD_READ_OUTPUT_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_READ_OUTPUT_CNF_T			
ulRemAdd	UINT32	0...126	Remote address of PROFIBUS DP slave station from which output data have been requested.
abData	UINT8[]		Area for output data

Table 125: PROFIBUS_FSPMM2_CMD_READ_OUTPUT_CNF – Confirmation of Read Output

6.2.10 PROFIBUS_FSPMM2_CMD_GET_CFG_REQ/CNF – Get Configuration Command

This packet is used to request configuration information from a PROFIBUS DP slave.

The parameter `ulRemAdd` contains the PROFIBUS address of the slave station from which configuration data should be transferred.

The confirmation packet contains this address again and additionally the requested configuration data. These data are returned in parameter `abData[]` and sometimes denominated as 'real configuration data'. The 'real configuration data' originate from the slave. These data are structured in the way described in section *Configuration of Inputs and Outputs at Slaves* on page 19. Their length is limited to 244 bytes by the PROFIBUS specification.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_GET_CFG_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
} PROFIBUS_FSPMM2_GET_CFG_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_GET_CFG_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_GET_CFG_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_GET_CFG_REQ_T;

#define PROFIBUS_FSPMM2_GET_CFG_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_GET_CFG_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_GET_CFG_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4414	PROFIBUS_FSPMM2_CMD_GET_CFG_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_GET_CFG_REQ_T			
ulRemAdd	UINT32	0...126	Remote address of PROFIBUS DP slave station from which configuration information shall be requested.

Table 126: PROFIBUS_FSPMM2_CMD_GET_CFG_REQ – Get Configuration

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_GET_CFG_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
} PROFIBUS_FSPMM2_GET_CFG_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_GET_CFG_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_GET_CFG_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_GET_CFG_CNF_T;

#define PROFIBUS_FSPMM2_GET_CFG_CNF_SIZE (sizeof(PROFIBUS_FSPMM2_GET_CFG_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_GET_CFG_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	4+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4415	PROFIBUS_FSPMM2_CMD_GET_CFG_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_GET_CFG_CNF_T			
ulRemAdd	UINT32	0...126	Remote address of PROFIBUS DP slave station from which configuration information shall be requested.
abData[]	UINT8[PROFIBUS_FSPM_MAX_IO_DATA_LEN]		Data area containing configuration information

Table 127: PROFIBUS_FSPMM2_CMD_GET_CFG_CNF – Confirmation of Get Configuration

6.2.11 PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ/CNF – Set Slave Address (DP V1 Class2)

This packet allows changing the address of a single PROFIBUS DP slave by sending a packet to it while the PROFIBUS system fully remains in operation. However, this function is not supported by all slaves.

It is not necessary to address the slave only by its PROFIBUS address (which is specified in parameter `ulRemAdd` as usual) which was valid up to now but it is intended to be changed, but additionally an identification number is necessary to securely identify the slave to be addressed. This identification number is a 16-bit value to be specified in parameter `usIdentNumber`. This identification number characterizes the device type of a PROFIBUS DP V1 device and is assigned exclusively by the PROFIBUS International organization.

Besides the new slave address itself which is transferred in parameter `bNewSlaveAdd`, this service may include also further slave specific data which are usually stored in a permanent memory at the slave. If necessary, these data are transferred in the field `abRemSlaveData[]`. A structure of these data cannot be given as this depends on how your slave interprets these data, so check the slave's documentation for more information on this topic.

The parameter `bNoAddChg` indicates whether further changes of the address of this slave shall still be allowed (`bNoAddChg = 0`) or be disabled in future (`bNoAddChg = 1`).

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_SET_SLAVE_ADD_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  bNewSlaveAdd;
  TLR_UINT16 usIdentNumber;
  TLR_UINT8  bNoAddChg;
  TLR_UINT8  abRemSlaveData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
} PROFIBUS_FSPMM2_SET_SLAVE_ADD_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_SET_SLAVE_ADD_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_SET_SLAVE_ADD_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_SET_SLAVE_ADD_REQ_T;

#define PROFIBUS_FSPMM2_SET_SLAVE_ADD_REQ_SIZE
(sizeof(PROFIBUS_FSPMM2_SET_SLAVE_ADD_REQ_T)-PROFIBUS_FSPM_MAX_IO_DATA_LEN)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_SET_SLAVE_ADD_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4416	PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_SET_SLAVE_ADD_REQ_T			
ulRemAdd	UINT32	0...126	Remote address of PROFIBUS DP slave station whose address is intended be changed.
bNewSlaveAdd	UINT8	0...125	The value of the new address the slave should have in future.
usIdentNumber	UINT16	Depends on slave	Identification number of the slave whose address is to be changed.
bNoAddChg	UINT8	0,1	1: No further address changes possible. Address is locked for this slave. 0: Address changes still allowed in future
abRemSlaveData	UINT8[]		Area for data to be sent to the slave and to be stored in a permanent memory there.

Table 128: PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ – Set Slave Address

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_SET_SLAVE_ADD_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  bNewSlaveAdd;
  TLR_UINT16 usIdentNumber;
  TLR_UINT8  bNoAddChg;
} PROFIBUS_FSPMM2_SET_SLAVE_ADD_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_SET_SLAVE_ADD_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_SET_SLAVE_ADD_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_SET_SLAVE_ADD_CNF_T;

#define PROFIBUS_FSPMM2_SET_SLAVE_ADD_CNF_SIZE
(sizeof(PROFIBUS_FSPMM2_SET_SLAVE_ADD_CNF_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_SET_SLAVE_ADD_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	8	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.3 Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4417	PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_SET_SLAVE_ADD_CNF_T			
ulRemAdd	UINT32	0...126	Former remote address of PROFIBUS DP slave station. If an error occurred this remote address will remain in effect.
bNewSlaveAdd	UINT8		The value of the new address the slave will have in future.
usIdentNumber	UINT16		Identification number of the slave whose address is to be changed.
bNoAddChg	UINT8		See request packet for explanation.

Table 129: PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_CNF – Confirmation of Set Slave Address

6.2.12 PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ/CNF – Get Master Diagnosis

This packet allows to request a diagnosis from a PROFIBUS DP V1 Master Class1.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  bIdentifier;
  TLR_UINT8  abReserved[3];
} PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_GET_MASTER_DIAG_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_GET_MASTER_DIAG_REQ_T;

#define PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_SIZE
(sizeof(PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_GET_MASTER_DIAG_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4418	PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_T			
ulRemAdd	UINT32	0...125	Remote address of PROFIBUS DP Master station from which diagnostic data shall be requested.
bIdentifier	UINT8	0…255	Identifier
abReserved[3]	UINT8 []		Reserved

Table 130: PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ – Get Master Diagnosis

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_GET_MASTER_DIAG_CNF_Ttag
{
  TLR_UINT32 ulRemAdd;
  TLR_UINT8  bIdentifier;
  TLR_UINT8  abReserved[3];
  TLR_UINT8  abData[PROFIBUS_FSPM_MAX_IO_DATA_LEN];
} PROFIBUS_FSPMM2_GET_MASTER_DIAG_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_GET_MASTER_DIAG_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_GET_MASTER_DIAG_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_GET_MASTER_DIAG_CNF_T;

#define PROFIBUS_FSPMM2_GET_MASTER_DIAG_CNF_SIZE
(sizeof(PROFIBUS_FSPMM2_GET_MASTER_DIAG_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_GET_MASTER_DIAG_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	8+n	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4419	PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_GET_MASTER_DIAG_CNF_T			
ulRemAdd	UINT32	0...125	Remote address
bIdentifier	UINT8	0…255	Identifier
abReserved[3]	UINT8[]		Reserved
abData	UINT8[]		Diagnostic data

Table 131: PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_CNF – Confirmation of Get Master Diagnosis

6.2.13 PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ/CNF – Live List Command

Using this packet it is possible to request a live list, i.e. a list of stations participating in the PROFIBUS DP network at the current time. In order to request a live list you do not have to specify any parameters while the confirmation packet contains the requested live list within the field `abLiveList[]`. The length of the live list is limited to 128 bytes.

For each remote address within the PROFIBUS network the byte with the corresponding index value (running from 0 to 126) within the `abLiveList[]` field may be set to one of the following values:

Coding of Live List Bytes

Value	Meaning
3	Master (active)
1	Slave (passive)
0	Not present

Table 132: Coding of Live List Bytes

The contents of the `abLiveList[]` field is of course only valid if the `ulSta` parameter is equal to 0.

Packet Structure Reference

```
#define PROFIBUS_FSPMM2_LIVE_LIST_SIZE 128
typedef struct PROFIBUS_FSPMM2_PACKET_LIVE_LIST_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_FSPMM2_PACKET_LIVE_LIST_REQ_T;

#define PROFIBUS_FSPMM2_LIVE_LIST_REQ_SIZE     (0)
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_LIVE_LIST_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4430	PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 133: PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ - – Live List Command

Packet Structure Reference

```
#define PROFIBUS_FSPMM2_LIVE_LIST_SIZE 128
typedef struct  PROFIBUS_FSPMM2_LIVE_LIST_CNF_Ttag
{
  TLR_UINT8  abLiveList[PROFIBUS_FSPMM2_LIVE_LIST_SIZE];
} PROFIBUS_FSPMM2_LIVE_LIST_CNF_T;

typedef struct PROFIBUS_FSPMM2_PACKET_LIVE_LIST_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_LIVE_LIST_CNF_T tData;
} PROFIBUS_FSPMM2_PACKET_LIVE_LIST_CNF_T;

#define PROFIBUS_FSPMM2_LIVE_LIST_CNF_NEG_SIZE (0)
#define PROFIBUS_FSPMM2_LIVE_LIST_CNF_POS_SIZE (sizeof(PROFIBUS_FSPMM2_LIVE_LIST_CNF_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_LIVE_LIST_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	128	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4431	PROFIBUS_FSPMM2_CMD_LIVE_LIST_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_LIVE_LIST_CNF_T			
abLiveList	UINT8[]		Live list data

Table 134: PROFIBUS_FSPMM2_CMD_LIVE_LIST_CNF – Confirmation of Live List Request

6.2.14 PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication

This indication signals that a slave in connection with the Hilscher PROFIBUS DP device has requested an abort of the connection with the Hilscher device acting as PROFIBUS DP Master Class2. The reason why the connection should be aborted is specified in the reason code.

A PROFIBUS_FSPMM2_CMD_ABORT_IND indication at the host side can be received at any time as long as the connection is in established state. Getting an abort does not implicitly mean that the connection is closed, it only serves to indicate to the host that the connection has been aborted, either locally or remote controlled. But each abort indication will close the connection afterwards automatically so that the host has to wait for the PROFIBUS_FSPMM2_CMD_CLOSED_IND indication coming from the device afterwards also. Only the PROFIBUS_FSPMM2_CMD_CLOSED_IND declares the connection as closed and only this indication can be used as a trigger to initialize the connection again by the PROFIBUS_FSPMM2_CMD_INITIATE_REQ packet.

An abort can either be forced by the remote slave station (bLocallyGenerated=0) or can be generated locally (bLocallyGenerated=1), for example in case of communication errors. The reason why (bReasonCode) is passed over in the packet, too.

In detail the parameters have the following meaning:

1. The ulCRef parameter contains the communication reference in order to uniquely identify which connection has to be aborted.
2. The bLocallyGenerated parameter indicates whether the abort has been caused locally or remotely as described above.
3. The bSubnet parameter indicates the source of the abort, i.e.. whether it originates from the local or a remote subnet or no specific source information should be given. For the coding see *Table 115: Allowed Values of Subnet Variable*.
4. The bInstance parameter specifies the protocol instance causing the abort. Possible values are FDL, USER, MSAC_C2. For the coding see *Table 116: Instance Codes and their Meaning*.
5. The bReasonCode parameter is a 4 bit value (i.e. between 0 and 15) whose meaning depends from the instance and indicates the reason for the abort. For the coding see *Table 117: Possible Reason Codes and their Meaning*.
6. The usAbortDetail parameter contains additional information about the reason of the abort if available if not equal to 0. In case of a too small value for the send timeout (bInstance = MSAC_C2 and bReasonCode = ABT_STO) this parameter should contain the Send_Timeout value of the connection to be aborted according to the IEC 61158 Part 3/EN 50170 specification.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_ABORT_IND_Ttag
{
  TLR_UINT32 ulCRef;
  TLR_UINT8  bLocallyGenerated;
  TLR_UINT8  bSubnet;
  TLR_UINT8  bInstance;
  TLR_UINT8  bReasonCode;
  TLR_UINT16 usAbortDetail;
  TLR_UINT8  abReserved[2];
}PROFIBUS_FSPMM2_ABORT_IND_T;

typedef struct PROFIBUS_FSPMM2_PACKET_ABORT_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_ABORT_IND_T tData;
}PROFIBUS_FSPMM2_PACKET_ABORT_IND_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_ABORT_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	12	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4426	PROFIBUS_FSPMM2_CMD_ABORT_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_ABORT_IND_T			
ulCRef	UINT32		Connection reference
bLocallyGenerated	UINT8	0,1	Indicates the source of the abort according to this coding: 0: Remotely Generated 1: Locally Generated
bSubnet	UINT8	0...2, 3...255 are reserved	Subnet
bInstance	UINT8	0...255	Instance
bReasonCode	UINT8	0...255	Reason code explaining reason for intended abort of connection
usAbortDetail	UINT16	0...65535	More detailed code for reason of abort
abReserved[2]	UINT8[]		Reserved area

Table 135: PROFIBUS_FSPMM2_CMD_ABORT_IND - Abort Indication

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_ABORT_RES_Ttag
{
  TLR_UINT32  ulCRef;
}PROFIBUS_FSPMM2_ABORT_RES_T;

typedef struct PROFIBUS_FSPMM2_PACKET_ABORT_RES_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_ABORT_RES_T tData;
}PROFIBUS_FSPMM2_PACKET_ABORT_RES_T;
```

Packet Description

structure PROFIBUS_FSPMM2_CMD_ABORT_RES			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, unchanged
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4427	PROFIBUS_FSPMM2_CMD_ABORT_RES - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_ABORT_RES_T			
ulCRef	UINT32		Connection reference

Table 136: PROFIBUS_FSPMM2_CMD_ABORT_RES – Response to Abort Indication

6.2.15 PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication/Response

This indication is sent from the FSPMM2-task to the AP-task in order to declare a previously initialized Class2 connection as closed. No further messages like PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication must be awaited for any longer for this now closed connection.

The only parameter is the communication reference identifying the now closed connection.

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_CLOSED_IND_Ttag
{
  TLR_UINT32 ulCRef;
}PROFIBUS_FSPMM2_CLOSED_IND_T;

typedef struct PROFIBUS_FSPMM2_PACKET_CLOSED_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_CLOSED_IND_T tData;
}PROFIBUS_FSPMM2_PACKET_CLOSED_IND_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_CLOSED_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4428	PROFIBUS_FSPMM2_CMD_CLOSED_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_CLOSED_IND_T			
ulCRef	UINT32		Reference of the closed connection

Table 137: PROFIBUS_FSPMM2_CMD_CLOSED_IND - Closed Indication

Packet Structure Reference

```
typedef struct  PROFIBUS_FSPMM2_CLOSED_RES_Ttag
{
  TLR_UINT32  ulCRef;
}PROFIBUS_FSPMM2_CLOSED_RES_T;

typedef struct PROFIBUS_FSPMM2_PACKET_CLOSED_RES_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_CLOSED_RES_T tData;
}PROFIBUS_FSPMM2_PACKET_CLOSED_RES_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_CLOSED_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, unchanged
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4429	PROFIBUS_FSPMM2_CMD_CLOSED_RES - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_CLOSED_RES_T			
ulCRef	UINT32		Connection reference

Table 138: PROFIBUS_FSPMM2_CMD_CLOSED_RES –Response to Closed Indication

6.2.16 PROFIBUS_FSPMM2_CMD_RESET_CONN_REQ/CNF – Reset Connection Command

This packet allows to selectively reset the DPV 1 Class2 connection to one single PROFIBUS-DP Slave which is selected by the parameter `ulRemAddr`.

Packet Structure Reference

```
/*****************************************************************************
 ** RESET REQ PACKET
 */
typedef struct PROFIBUS_FSPMM2_RESET_CONN_REQ_Ttag
{
  TLR_UINT32 ulRemAddr;
} PROFIBUS_FSPMM2_RESET_CONN_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_RESET_CONN_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_FSPMM2_RESET_CONN_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_RESET_CONN_REQ_T;

#define PROFIBUS_FSPMM2_RESET_CONN_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_RESET_CONN_REQ_T))
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_RESET_CONN_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulFSPMM2Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS2Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4432	PROFIBUS_FSPMM2_CMD_RESET_CONN_REQ- Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_FSPMM2_RESET_CONN_REQ_T			
ulRemAddr	UINT32	0...126	Remote address of PROFIBUS DP slave station to which the connection should be reset

Table 139: PROFIBUS_FSPMM2_CMD_RESET_CONN_REQ – Reset Connection Request

Packet Structure Reference

```
/*****************************************************************************
 ** RESET CNF PACKET
 */
#define PROFIBUS_FSPMM2_RESET_CONN_CNF_SIZE (0)

typedef struct PROFIBUS_FSPMM2_PACKET_RESET_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}PROFIBUS_FSPMM2_PACKET_RESET_CONN_CNF_T;
```

Packet Description

structure PROFIBUS_FSPMM2_PACKET_RESET_CONN_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM2Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulFSPMM2d	Source end point identifier, unchanged
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section 8.3 *Error Codes of the FSPMM2-Task*
ulCmd	UINT32	0x4433	PROFIBUS_FSPMM2_CMD_RESET_CONN_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change

Table 140: PROFIBUS_FSPMM2_CMD_RESET_CONN_CNF – Confirmation of Reset Connection Request

6.3 The DL-Task

At the PROFIBUS DL task, the FDL functionality is implemented.

To get the handle of the process queue of the DL-Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII queue name	Description
"PB_DL_QUE"	Name of the DL-Task process queue

Table 141: FSPMM-Task process queue

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the DL-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the DL-Task.

In detail, the following functionality is provided by the `PROFIBUS DL`-Task:

Overview over Packets of the `PROFIBUS DL` -Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.3.1	PROFIBUS_DL_CMD_START_DLE_REQ/CNF – Starts the DL-Layer	0x102/ 0x103	200
6.3.2	PROFIBUS_DL_CMD_STOP_DLE_REQ/CNF – Stop DL Layer	0x104/ 0x105	202
6.3.3	PROFIBUS_DL_CMD_DATA_ACK_REQ/CNF - Send SDA Service	0x106/ 0x107	204
6.3.4	PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service	0x108	209
6.3.5	PROFIBUS_DL_CMD_DATA_REQ/CNF - Send SDN Service	0x10A/ 0x10B	211
6.3.6	PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication	0x10C	215
6.3.7	PROFIBUS_DL_CMD_DATA_REPLY_REQ/CNF - Send SRD Service	0x10E/ 0x10F	217
6.3.8	PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication	0x110	221
6.3.9	PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ/CNF - Reply Update Service	0x112/ 0x113	224
6.3.10	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request	0x120/ 0x121	228
6.3.11	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ/CNF - Activate an SAP for Responder Functionality	0x122/ 0x123	233
6.3.12	PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request	0x128/ 0x129	238
6.3.13	PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set	0x126/ 0x127	240
6.3.14	PROFIBUS_DL_CMD_SET_VALUE_REQ/CNF - Set Value Service	0x12A/ 0x12B	244
6.3.15	PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ/CNF – Obtain FDL Status	0x132/ 0x133	247
6.3.16	PROFIBUS_DL_CMD_GET_LMS_REQ/CNF - Get List of active Stations	0x0134/ 0x0135	249
6.3.17	PROFIBUS_DL_CMD_REGISTER_LMS_REQ/CNF - Register an Application to receive LMS Change Indications	0x0136/ 0x0137	252
6.3.18	PROFIBUS_DL_CMD_LMS_CHANGED_IND - Indication for Change in LMS	0x0138	255

Overview over Packets of the PROFIBUS DL -Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.3.19	PROFIBUS_DL_CMD_GET_LL_REQ/CNF – Get the Life List (List of Active or Passive Stations)	0x0130/ 0x0131	257

Table 142: Overview over the Packets of the PROFIBUS DL -Task of the PROFIBUS DP-Master Protocol Stack

6.3.1 PROFIBUS_DL_CMD_START_DLE_REQ/CNF – Starts the DL-Layer

This packet should be sent to start the DL Layer with the current bus parameter settings.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_PACKET_CMD_START_DLE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_CMD_START_DLE_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_CMD_START_DLE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 “Error Codes of the DL-Task”
ulCmd	UINT32	0x102	PROFIBUS_DL_CMD_START_DLE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

Table 143: PROFIBUS_DL_CMD_START_DLE_REQ – Start the DL-Layer Request

Packet Structure Reference

```
typedef struct PROFIBUS_DL_PACKET_CMD_START_DLE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_CMD_START_DLE_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_CMD_START_DLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 Packets for Configuration during running system.
ulCmd	UINT32	0x103	PROFIBUS_DL_CMD_START_DLE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 144: PROFIBUS_DL_CMD_START_DLE_CNF – Confirmation of Start the DL-Layer Request

6.3.2 PROFIBUS_DL_CMD_STOP_DLE_REQ/CNF – Stop DL Layer

This packet shall be sent to stop the DL Layer.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_PACKET_CMD_STOP_DLE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_CMD_STOP_DLE_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_CMD_STOP_DLE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 “Error Codes of the DL-Task”.
ulCmd	UINT32	0x104	PROFIBUS_DL_CMD_STOP_DLE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

Table 145: PROFIBUS_DL_CMD_STOP_DLE_REQ– Stop DL Layer Request

Packet Structure Reference

```
typedef struct PROFIBUS_DL_PACKET_CMD_STOP_DLE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_CMD_STOP_DLE_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_CMD_STOP_DLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 “Packets for Configuration during running system“.
ulCmd	UINT32	0x105	PROFIBUS_DL_CMD_STOP_DLE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 146: PROFIBUS_DL_CMD_STOP_DLE_CNF – Confirmation of Stop DL Layer Request

6.3.3 PROFIBUS_DL_CMD_DATA_ACK_REQ/CNF - Send SDA Service

This packet provides the SDA (Send Data with Acknowledge) service for acknowledged connectionless data transfer with immediate response. It allows transparently sending an SDA frame with variable data length to the PROFIBUS network. At the remote station, the user data (which is contained in the Service Data Unit represented by the variable `abSDU[]`) is delivered to the user. The length of the Service Data Unit is limited to at most 246 bytes

The local confirmation message of this command informs about the receipt or non-receipt of the user data in the remote station. Within the confirmation packet, a value of `ulSta` equal to the following indicates successful receipt of the SDA message:

- `TLR_S_OK (0x00000000)`
- `TLR_E_PROFIBUS_DL_ACK_NR (0xC0060083L)`
- `TLR_E_PROFIBUS_DL_ACK_DH (0xC0060086L)`
- `TLR_E_PROFIBUS_DL_ACK_DL (0xC0060087L)`

All other values indicate the failure of the SDA service.

Packet Structure Reference

```
  typedef struct PROFIBUS_DL_DATA_ACK_REQ_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 abPad[4];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_ACK_REQ_T;

  #define PROFIBUS_DL_DATA_ACK_REQ_SIZE (sizeof(PROFIBUS_DL_DATA_ACK_REQ_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

  /* Request-Packet for acknowledged connectionless data transfer */
  typedef struct  PROFIBUS_DL_PACKET_DATA_ACK_REQ_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_ACK_REQ_T tData;    /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DATA_ACK_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_ACK_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 Error Codes of the DL-Task.
ulCmd	UINT32	0x106	PROFIBUS_DL_CMD_DATA_ACK_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_ACK_REQ_T			
bSrvCls	UINT8	0,1	Service Class (indicating the priority)
bDstAddr	UINT8	0-126	Destination address within the PROFIBUS network **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Destination Service Access Point
bSrcSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Source Service Access Point
abPad	UINT8[4]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 147: PROFIBUS_DL_CMD_DATA_ACK_REQ - Send SDA Service

Additional explanations:

1. The parameter `bSrvCls` defines the FDL priority for the data transfer. Two priorities are possible in this context:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0
   - PROFIBUS_DL_SERVICE_CLASS_HIGH = 1
2. The parameter `bDstAddr` defines the FDL address of the remote station. For this parameter, the value 127 is not permissible as remote address because it signifies the global address used for broadcasts.
3. The parameter `bDsap` defines the service access point of the remote user. The value 64 represents the NIL SAP.
4. The parameter `bSsap` defines the service access point of the local user. The value 64 again represents the NIL SAP.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DATA_ACK_CNF_Ttag {
  TLR_UINT8 bSrvCls;    /* Service Class */
  TLR_UINT8 bDstAddr;   /* Destination address */
  TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
  TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
} PROFIBUS_DL_DATA_ACK_CNF_T;

/* Confirmation-Packet for acknowledged connectionless data transfer */
typedef struct  PROFIBUS_DL_PACKET_DATA_ACK_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_DATA_ACK_CNF_T tData;    /* Packet Data Unit */
} PROFIBUS_DL_PACKET_DATA_ACK_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_ACK_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 “Packets for Configuration during running system”.
ulCmd	UINT32	0x107	PROFIBUS_DL_CMD_DATA_ACK_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_DATA_ACK_CNF_T			
bSrvCls	UINT8	0,1	Service Class
bDstAddr	UINT8	0-126	Destination address Note: Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-62, 64 = NIL SAP	Destination Service Access Point
bSrcSAPIdx	UINT8	0-62, 64 = NIL SAP	Source Service Access Point

Table 148: PROFIBUS_DL_CMD_DATA_ACK_CNF – Confirmation of Send SDA Service

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_PROFIBUS_DL_ACK_UE (0xC0060080L)	The remote station the service has been sent to indicates a User Error as service acknowledgement.
TLR_E_PROFIBUS_DL_ACK_RR (0xC0060081L)	Some resource is not available at the remote station as indicated by the remote station, for instance the slave has no left buffer space for the requested service. Action: Increase buffer space or reduce amount of required buffer space.
TLR_E_PROFIBUS_DL_ACK_RS (0xC0060082L	The remote station the service has been sent to indicates a Service Access Point Error as service acknowledgement.
TLR_E_PROFIBUS_DL_ACK_NR (0xC0060083L)	The remote station the service has been sent to confirms its positive reception but has no data to confirm.
TLR_E_PROFIBUS_DL_ACK_RDH (0xC0060084L)	The remote station the service has been sent to, confirms its reception negatively but has returned low priority data in the response.
TLR_E_PROFIBUS_DL_ACK_RDL (0xC0060085L)	The remote station the service has been sent to, confirms its reception negatively but has returned high priority data in the response.
TLR_E_PROFIBUS_DL_ACK_DH (0xC0060086L)	The remote station the service has been sent to, confirms its reception positively and has returned low priority data in the response.
TLR_E_PROFIBUS_DL_ACK_DL (0xC0060087L)	The remote station the service has been sent to, confirms its reception positively and has returned high priority data in the response.
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	The remote station the service has been sent to shows no or no plausible reaction at all.
TLR_E_PROFIBUS_DL_ACK_UNKNOWN (0xC0060089L)	The remote station the service has been sent has returned an unknown acknowledgement code.

Table 149: PROFIBUS_DL_CMD_DATA_ACK_CNF - Packet Status/Error

For the following causes of failures some hints can be given:

Error Definition / (Value)	Error source	Description of problem and necessary actions
TLR_E_PROFIBUS_DL_ACK_LR (0xC006008BL)	Slave	The local resources needed to execute the requested service are not available or not sufficient for instance the slave has no left buffer space for the requested service. Action: Increase buffer space or reduce amount of required buffer space.
TLR_E_PROFIBUS_DL_ACK_LS (0xC006008AL)	Slave	The requested function of master is not activated within the slave, i.e. the slave does not support the service which had been requested. Action: If possible, use another slave providing the requested functionality.
TLR_E_PROFIBUS_DL_ACK_NR (0xC0060083L)	Slave	No answer-data available, as the slave did not sent back any data. Action: Check slave for correct operation.
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	Slave	No response of the station at all Action: Check for correct network wiring, also check the bus address of slave or baud rate support.
TLR_E_PROFIBUS_DL_ACK_DS (0xC006008CL)	Network in general	The master is not included into the logical token ring. Action: Check master DP-Address or highest-station-address of other masters Examine bus wiring to avoid bus short circuits.

Table 150: Reported Errors, their Sources and Possible Actions

6.3.4 PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service

This indication is transferred from the remote FDL controller to the local host, when an SDA (Send Data with Acknowledge) frame has been received from an initiator on the PROFIBUS network. The reception of the indication message needs no acknowledgment message to the local FDL.

The data of the message is contained in the Service Data Unit, represented by the field `abSDU[]`. The length of the Service Data Unit is limited to at most 246 bytes

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DATA_ACK_IND_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcAddr;   /* Source address */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 abPad[3];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_ACK_IND_T;

  #define PROFIBUS_DL_DATA_ACK_IND_SIZE (sizeof(PROFIBUS_DL_DATA_ACK_IND_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

 /* Indication Packet for acknowledged connectionless data transfer */
  typedef struct  PROFIBUS_DL_PACKET_DATA_ACK_IND_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_ACK_IND_T tData;
  } PROFIBUS_DL_PACKET_DATA_ACK_IND_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_ACK_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 Error Codes of the DL-Task.
ulCmd	UINT32	0x108	PROFIBUS_DL_CMD_DATA_ACK_IND - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_ACK_IND_T			
bSrvCls	UINT8	0,1	Service Class
bDstAddr	UINT8	0-126	Destination address **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Destination Service Access Point
bSrcAddr	UINT8	0-126	Source address **Note:** Mask out the highest bit when working with bSrcAddr in order to avoid disturbances as this bit stores a separate information.
bSrcSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Source Service Access Point
abPad	UINT8[3]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 151: PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service

Additional explanations

1. The parameter `bSrcSAPIdx` and `bDstSAPIdx` specify the source and destination Service Access Points of the received SDA frame.
2. The parameter `bDstAddr` defines the FDL address of the remote station, to which this service has been sent.
3. The parameter `bSrcAddr` defines the FDL address of the remote station, from which this service has been sent.
4. The parameter `bSrvCls` specifies the FDL priority of the received SDA frame. Two priorities are possible in this context:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0
   - PROFIBUS_DL_SERVICE_CLASS_HIGH = 1

6.3.5 PROFIBUS_DL_CMD_DATA_REQ/CNF - Send SDN Service

This packet provides the SDN (Send Data with no Acknowledge) service for unacknowledged connectionless data transfer to one or more stations in the PROFIBUS network. It allows transparently sending a SDN (Send Data with no Acknowledge) frame to the PROFIBUS network with variable data length to a single remote station, to many remote stations (multicast) or to all remote stations (broadcast) at the same time. At the remote station(s), the user data (which is contained in the Service Data Unit represented by the variable `abSDU[]`) is delivered to the user. The length of the Service Data Unit is limited to at most 246 bytes

The local confirmation message of this command informs just the end of the transfer, but not about successful reception of the data. Within the confirmation packet, a value of `ulSta` equal to `TLR_S_OK (0x00000000)` indicates successful receipt of the SDN message: All other values indicate the failure of the SDN service.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DATA_REQ_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 abPad[4];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_REQ_T;

  #define PROFIBUS_DL_DATA_REQ_SIZE (sizeof(PROFIBUS_DL_DATA_REQ_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

  /* Request-Packet for unacknowledged connectionless data transfer */
  typedef struct  PROFIBUS_DL_PACKET_DATA_REQ_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_REQ_T tData;  /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DATA_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 Error Codes of the DL-Task.
ulCmd	UINT32	0x10A	PROFIBUS_DL_CMD_DATA_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_REQ_T			
bSrvCls	UINT8	0,1	Service Class (indicating the priority)
bDstAddr	UINT8	0-126, 127 (= Broadcast)	Destination address within the PROFIBUS network **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-63, 64 = NIL SAP	Destination Service Access Point
bSrcSAPIdx	UINT8	0-62, 64 = NIL SAP	Source Service Access Point
abPad	UINT8[4]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 152: PROFIBUS_DL_CMD_DATA_REQ - Send SDN Service Request

Additional explanations:

1. The parameter `bSrvCls` defines the FDL priority for the data transfer. Two priorities are possible in this context:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0
   - PROFIBUS_DL_SERVICE_CLASS_HIGH = 1
2. The parameter `bDstAddr` defines the FDL address of the remote station. The allowed range for this parameter extends from `0` to `126.`
3. The parameter `bDstSAPIdx` defines the service access point of the remote user. The value 64 represents the NIL SAP.
4. The parameter `bSrcSAPIdx` defines the service access point of the local user. The allowed range for this parameter extends from `0` to `62.` A value `bSrcSAPIdx` of 63 is not permitted in this command. The value 64 again represents the NIL SAP.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DATA_CNF_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
  } PROFIBUS_DL_DATA_CNF_T;

  /* Confirmation-Packet for unacknowledged connectionless data transfer */
  typedef struct  PROFIBUS_DL_PACKET_DATA_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_CNF_T tData;    /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DATA_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	4	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See Table 154: PROFIBUS_DL_CMD_DATA_CNF - Packet Status/Error
ulCmd	UINT32	0x10B	PROFIBUS_DL_CMD_DATA_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_DATA_CNF_T			
bSrcCls	UINT8	0,1	Service Class
bDstAddr	UINT8	0-126, 127 (= Broadcast)	Destination address Note: Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-63, 64 = NIL SAP	Destination Service Access Point
bSrcSAPIdx	UINT8	0-62, 64 = NIL SAP	Source Service Access Point

Table 153: PROFIBUS_DL_CMD_DATA_CNF – Confirmation of Send SDN Service Request

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_PROFIBUS_DL_ACK_RR (0xC0060081L)	The remote station the service has been sent to indicates a Resource Error as service acknowledgment
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	The remote station the service has been sent to shows no or no plausible reaction at all.
TLR_E_PROFIBUS_DL_ACK_LS (0xC006008AL)	The requested service is not activated within the local SAP configuration.
TLR_E_PROFIBUS_DL_ACK_LR (0xC006008BL)	The local resources needed to execute the requested service are not available or not sufficient.
TLR_E_PROFIBUS_DL_ACK_DS (0xC006008CL)	Master not into the logical token ring

Table 154: PROFIBUS_DL_CMD_DATA_CNF - Packet Status/Error

For the following causes of failures some hints can be given:

Error Definition / (Value)	Error source	Description of problem and necessary actions
TLR_E_PROFIBUS_DL_ACK_RR (0xC0060081L)	Slave	Some resource is not available at the remote station as indicated by the remote station, for instance the slave has no left buffer space for the requested service. Action: Increase buffer space or reduce amount of required buffer space.
TLR_E_PROFIBUS_DL_ACK_DS (0xC006008CL)	Network in general	The master is not included into the logical token ring. Action: Check master DP-Address or highest-station-address of other masters Examine bus wiring to avoid bus short circuits.

Table 155: Reported Errors, their Sources and Possible Actions

6.3.6 PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication

This indication is passed from the remote FDL controller to the local host, when a SDN (Send Data with Acknowledge) frame has been received. The reception of the indication message needs no acknowledgment message to the local FDL.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DATA_IND_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcAddr;   /* Source address */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 abPad[3];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_IND_T;

   #define PROFIBUS_DL_DATA_IND_SIZE (sizeof(PROFIBUS_DL_DATA_IND_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

 /* Indication Packet for unacknowledged connectionless data transfer */
  typedef struct  PROFIBUS_DL_PACKET_DATA_IND_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_IND_T tData;
  } PROFIBUS_DL_PACKET_DATA_IND_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 Error Codes of the DL-Task
ulCmd	UINT32	0x10C	PROFIBUS_DL_CMD_DATA_IND - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

structure PROFIBUS_DL_DATA_IND_T			
bSrcCls	UINT8	0,1	Service Class
bDstAddr	UINT8	0-126, 127 (= Broadcast)	Destination address **Note:** Mask out the highest bit when working with bDstAddr to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-63, 64 (= NIL SAP)	Destination Service Access Point
bSrcAddr	UINT8	0-126	Source address **Note:** Mask out the highest bit when working with bSrcAddr to avoid disturbances as this bit stores a separate information.
bSrcSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Source Service Access Point
abPad	UINT8[3]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 156: PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication

Additional explanations

1. The parameter `bSrcSAPIdx` and `bDstSAPIdx` specify the source and destination service access points of the received SDN frame. The allowed range for this parameter extends from `0` to `62`. The value 64 is also allowed. It represents the NIL SAP. A value `bSrcSAPIdx` of 63 is not permissible in this command because it represents the global access.
2. The parameter `bSrcAddr` defines the FDL address of the local station. Values between 0 and 126 are allowed.
3. The parameter `bDstAddr` defines the FDL address of the remote station, which has sent this service. Values between 0 and 126 are allowed. 127 signifies a broadcast.
4. The parameter `bSrvCls` defines the FDL priority for the data transfer. Two priorities are possible in this context:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0
   - PROFIBUS_DL_SERVICE_CLASS_HIGH = 1

6.3.7 PROFIBUS_DL_CMD_DATA_REPLY_REQ/CNF - Send SRD Service

This packet shall be used to send the service SRD (Send and Request with Reply) for bidirectional connectionless data exchange. It allows transferring 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. The data of the response frame is sent back in the response message transparently.

This packet provides the SRD (Send and Request with Reply) service for data transfer with reply. It allows transferring data to a single remote station in the PROFIBUS network 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 user data (which is contained in the Service Data Unit represented by the variable `abSDU[]`) is delivered to the user. The length of the Service Data Unit is limited to at most 246 bytes.

Note: This service also allows to request data from the remote station without sending data to the remote station (in this case abSDU should be empty).

The local confirmation message of this command informs about the receipt or non-receipt of the user data in the remote station.

Packet Structure Reference

```
  typedef struct PROFIBUS_DL_DATA_REPLY_REQ_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 abPad[4];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_REPLY_REQ_T;

  #define PROFIBUS_DL_DATA_REPLY_REQ_SIZE (sizeof(PROFIBUS_DL_DATA_REPLY_REQ_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

  /* Request-Packet for two-way connectionless data exchange */
  typedef struct  PROFIBUS_DL_PACKET_DATA_REPLY_REQ_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_REPLY_REQ_T tData;    /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DATA_REPLY_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_REPLY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle, unchanged
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle, unchanged
ulDestId	UINT32	ulDL0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, unchanged
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 Error Codes of the DL-Task.
ulCmd	UINT32	0x10E	PROFIBUS_DL_CMD_DATA_REPLY_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_REPLY_REQ_T			
bSrvCls	UINT8	0,1	Service Class (indicating the priority)
bDstAddr	UINT8	0-126	Destination address within the PROFIBUS network **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-62, 64 = NIL SAP	Destination Service Access Point
bSrcSAPIdx	UINT8	0-62, 64 = NIL SAP	Source Service Access Point
abPad	UINT8[4]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 157: PROFIBUS_DL_CMD_DATA_REPLY_REQ - Send SRD Service Request

Additional explanations:

1. The parameter `bDstSAPIdx` defines the service access point of the remote user. It is not recommended to use the value 63 in this context. The value 64 represents the NIL SAP.
2. The parameter `bSrcSAPIdx` defines the service access point of the local user. It is not recommended to use the value 63 in this context. The value 64 represents the NIL SAP.
3. The parameter `bDstAddr` defines the FDL address of the remote station. The allowed range for this parameter extends from `0` to `126`.
4. The parameter `bSrvCls` defines the FDL priority for the data transfer. Two priorities are possible in this context:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0
   - PROFIBUS_DL_SERVICE_CLASS_HIGH = 1

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DATA_REPLY_CNF_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 abPad[4];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_REPLY_CNF_T;

  #define PROFIBUS_DL_DATA_REPLY_CNF_SIZE (sizeof(PROFIBUS_DL_DATA_REPLY_CNF_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

  /* Confirmation-Packet for two-way connectionless data exchange */
  typedef struct  PROFIBUS_DL_PACKET_DATA_REPLY_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_REPLY_CNF_T tData;    /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DATA_REPLY_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_REPLY_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See *Table 159: PROFIBUS_DL_CMD_DATA_REPLY_CNF - Packet Status/Error*
ulCmd	UINT32	0x10F	PROFIBUS_DL_CMD_DATA_REPLY_CNF- Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_DATA_REPLY_CNF_T			
bSrvCls	UINT8	0,1	Service Class
bDstAddr	UINT8	0-126	Destination address **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Destination Service Access Point
bSrcSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Source Service Access Point
abPad	UINT8[4]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 158: PROFIBUS_DL_CMD_DATA_REPLY_CNF – Confirmation of Send SRD Service Request

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_PROFIBUS_DL_ACK_DL (0xC0060087L)	The remote station the service has been sent to, confirms its reception positively and has returned low priority data in the response.
TLR_E_PROFIBUS_DL_ACK_RR (0xC0060081L)	The remote station the service has been sent to indicate a Resource Error as service acknowledgment
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	The remote station the service has been sent to shows no or no plausible reaction at all.
TLR_E_PROFIBUS_DL_ACK_LS (0xC006008AL)	The requested service is not activated within the local SAP configuration.
TLR_E_PROFIBUS_DL_ACK_LR (0xC006008BL)	The local resources needed to execute the requested service are not available or not sufficient.
TLR_E_PROFIBUS_DL_ACK_DS (0xC006008CL)	Master not into the logical token ring
TLR_E_PROFIBUS_DL_ACK_IV (0xC006008DL)	Invalid parameter detected in the requested service.

Table 159: PROFIBUS_DL_CMD_DATA_REPLY_CNF - Packet Status/Error

For the following possible causes of failures some hints can be given:

Error Definition / (Value)	Error source	Description of problem and necessary actions
TLR_E_PROFIBUS_DL_ACK_RR (0xC0060081L)	Slave	Some resource is not available at the remote station as indicated by the remote station, for instance the slave has no left buffer space for the requested service. Action: Increase buffer space or reduce amount of required buffer space.
TLR_E_PROFIBUS_DL_ACK_LR (0xC006008BL)	Slave	The local resources needed to execute the requested service are not available or not sufficient for instance the slave has no left buffer space for the requested service. Action: Increase buffer space or reduce amount of required buffer space.
TLR_E_PROFIBUS_DL_ACK_LS (0xC006008AL)	Slave	The requested function of master is not activated within the slave, i.e. the slave does not support the service which had been requested. Action: If possible, use another slave providing the requested functionality.
TLR_E_PROFIBUS_DL_ACK_DH (0xC0060086L)	No error	The slave has responded with data in a low priority telegram
TLR_E_PROFIBUS_DL_ACK_NR (0xC0060083L)	Slave	No answer-data available, the slave did not sent back any data. Action: Check slave for correct operation.
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	Slave	No response of the station at all Action: Check for correct network wiring, also check the bus address of slave or baud rate support.
TLR_E_PROFIBUS_DL_ACK_DS (0xC006008CL)	Network in general	The master is not included into the logical token ring. Action: Check master DP-Address or highest-station-address of other masters Examine bus wiring to avoid bus short circuits.

Table 160: Reported Errors, their Sources and Possible Actions

6.3.8 PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication

This indication is sent, when the service SRD (Send and Request with Reply) has been received from another station within the PROFIBUS network.

Packet Structure Reference

```
  typedef struct PROFIBUS_DL_DATA_REPLY_IND_Ttag {
    TLR_UINT8 bSrvCls;    /* Service Class */
    TLR_UINT8 bDstAddr;   /* Destination address */
    TLR_UINT8 bDstSAPIdx; /* Destination Service Access Point */
    TLR_UINT8 bSrcAddr;   /* Source address */
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 bUpdateStatus;   /* Indicates whether or not a response data has been
passed */
    TLR_UINT8 abPad[2];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
    TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
  } PROFIBUS_DL_DATA_REPLY_IND_T;

   #define PROFIBUS_DL_DATA_REPLY_IND_SIZE (sizeof(PROFIBUS_DL_DATA_REPLY_IND_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

 /* Indication Packet for acknowledged connectionless data transfer */
  typedef struct  PROFIBUS_DL_PACKET_DATA_REPLY_IND_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DATA_REPLY_IND_T tData;
  } PROFIBUS_DL_PACKET_DATA_REPLY_IND_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DATA_REPLY_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	8 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See section 8.4 Error Codes of the DL-Task.
ulCmd	UINT32	0x110	PROFIBUS_DL_CMD_DATA_REPLY_IND - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_REPLY_IND_T			
bSrvCls	UINT8	0,1	Service Class
bDstAddr	UINT8	0-126	Destination Address **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bDstSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Destination Service Access Point
bSrcAddr	UINT8	0-126	Source address **Note:** Mask out the highest bit when working with bDstAddr in order to avoid disturbances as this bit stores a separate information.
bSrcSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Source Service Access Point
bUpdateStatus	UINT8	0,1	Indicates whether or not response data has been passed
abPad	UINT8[2]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 161: PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication

Additional explanations:

1. The parameters `bSrcSAPIdx` and `bDstSAPIdx` specify the source and destination service access points of the received SRD frame. The allowed range for these parameters extends from `0` to `62`. The value 64 is also allowed. It represents the NIL SAP. However, a value `bSrcSAPIdx` of 63 is not permitted in this command because it represents the global access.
2. The parameter `bSrcAddr` defines the FDL address of the local station. Values between 0 and 126 are allowed.
3. The parameter `bDstAddr` defines the FDL address of the remote station, which has sent this service. Values between 0 and 126 are allowed.
4. The parameter `bSrvCls` specifies the FDL priority of the received SRD frame. Two priorities are possible in this context:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0
   - PROFIBUS_DL_SERVICE_CLASS_HIGH =1
5. The parameter `bUpdateStatus` indicates whether or not the response data have been passed to the local FDL controller. The response data can be set up to with the reply update service (`PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ`).

   The following table shows the allowed values of the parameter `bUpdateStatus` and their meanings:

Name	Value	Meaning
PROFIBUS_DL_UPDATE_STATUS_NO	0x00	No data transmitted in response
PROFIBUS_DL_UPDATE_STATUS_LO	0x01	Low priority data transmitted in response
PROFIBUS_DL_UPDATE_STATUS_HI	0x02	High priority data transmitted in response

Table 162: Allowed Values and their Meanings for Variable `bUpdateStatus`

6.3.9 PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ/CNF - Reply Update Service

The service is used to update the reply buffer of the SRD service (Send and Request with Reply).

The host is responsible for valid data in the device if any data should be requested from a remote station issuing an SRD or MSRD service request. By this command, the host may initiate the loading of this data to a specific SAP. Upon completion of loading the data area, the device informs the host by the confirmation message.

This packet is only appropriate if the intended role in service (value of the `bRoleInService` parameter) is not “Initiator” (`PROFIBUS_DL_SERVICE_ROLE_INITIATOR`). In this special case you are recommended to use the packet “PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request” described in the previous section.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_REPLY_UPDATE_REQ_Ttag {
  TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
  TLR_UINT8 bSrvCls;    /* Service Class */
  TLR_UINT8 bTransmitStrategy;   /* One shot or multiple transmissions */
  TLR_UINT8 abPad[1];   /* Padding needed to bring SDU to a DWORD boundary and to right
position */
  TLR_UINT8 abSDU[PROFIBUS_DL_MAX_DLPDU_SIZE]; /* Service Data Unit */
} PROFIBUS_DL_REPLY_UPDATE_REQ_T;

#define PROFIBUS_DL_REPLY_UPDATE_REQ_SIZE (sizeof(PROFIBUS_DL_REPLY_UPDATE_REQ_T)-
PROFIBUS_DL_MAX_DLPDU_SIZE)

/* Request-Packet for updating the reply update buffer of a two-way connectionless data
exchange */
typedef struct  PROFIBUS_DL_PACKET_REPLY_UPDATE_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_REPLY_UPDATE_REQ_T tData;    /* Packet Data Unit */
} PROFIBUS_DL_PACKET_REPLY_UPDATE_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_REPLY_UPDATE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	4 + n	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 Packets for Configuration during running system.
ulCmd	UINT32	0x112	PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_REPLY_UPDATE_REQ_T			
bSrcSAPIdx	UINT8	0-62, 64 (= NIL SAP)	Source Service Access Point
bSrvCls	UINT8	0,1	Service Class
bTransmitStrategy	UINT8	0,1	One shot or multiple transmissions
abPad	UINT8[1]	0	Padding bytes
abSDU	UINT8[]		Service Data Unit

Table 163: PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ - Reply Update Service

Additional explanations:

1. The parameter `bSrcSAPIdx` specifies the service access point of the remote user that carries out this request and which data area should be updated by the `abSDU`. A `bSrcSAPIdx` value of 63 is not permitted. A value of 64 represents the NIL SAP.
2. The parameter `bSrvCls` defines the FDL priority for the data transfer. Two priorities are possible:
   - PROFIBUS_DL_SERVICE_CLASS_LOW = 0 and
   - PROFIBUS_DL_SERVICE_CLASS_HIGH = 1.
3. The parameter `bTransmitStrategy` specifies whether the update is transmitted only once or many times, in the case of "multiple" the data area is transferred again for each subsequent SRD service. This is done by choosing between
   - PROFIBUS_DL_REPLY_UPDATE_SINGLE = 0 and
   - PROFIBUS_DL_REPLY_UPDATE_MULTIPLE = 1

Packet Structure Reference

```
typedef struct PROFIBUS_DL_REPLY_UPDATE_CNF_Ttag {
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point */
    TLR_UINT8 bSrvCls;    /* Service Class */
  } PROFIBUS_DL_REPLY_UPDATE_CNF_T;

  /* Confirmation-Packet for after updating the reply update buffer of a two-way
connectionless data exchange */
  typedef struct  PROFIBUS_DL_PACKET_REPLY_UPDATE_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_REPLY_UPDATE_CNF_T tData;     /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_REPLY_UPDATE_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_REPLY_UPDATE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	2	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 Packets for Configuration during running system.
ulCmd	UINT32	0x113	PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_REPLY_UPDATE_CNF_T			
bSrcSAPIdx	UINT8	0-62, 64 = NIL SAP	Source Service Access Point
bSrvCls	UINT8	0,1	Service Class

Table 164: PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_CNF – Confirmation of Reply Update Service

6.3.10 PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request

This packet allows configuring and activating a local service access point.

A service access point is a means for managing communication between different communication layers correctly, it is identified by a non-negative integer numeric value. You can find a detailed description in section 3.5.2 “*Management of Services Access Points*” on page 46 of this document.

This packet provides the host with the possibility to configure a local service access point (LSAP) for most of the individual FDL services. However, there is a special case requiring additional information if responding functionality is needed for services with integrated reply capabilities such as SRD and related ones. In this case the packet described here is not applicable, so there is a separate packet for configuring and activating the responder function for the Reply services (SRD, MSRD) that are activated by means of the RSAP activate service, see next the section

The parameter `bSrcSAPIdx` specifies the local LSAP that is to be activated and configured. The allowed values are 0 to 63 and NIL is represented by the value 64.

After the SAP has successfully been configured and activated, a confirmation packet containing the numerical value of the SAP (contained in variable `bSrcSAPIdx`) and the access protection mode will be sent back to the application. After that, the SAP may be used in FDL communication function such as the ones described in the sections 6.3.3, 6.3.5 or 6.3.7. If later on this SAP is not needed any longer, it may be deactivated using the packet “SAP Deactivate” described in section 3.5.2.4 „*SAP Deactivate*“ of this manual.

If a SAP should be configured for a responder service (i.e. the SRD or the MSRD service), this packet is only appropriate if the intended role in service (represented by the value of the `bRoleInService` parameter) is “Initiator” (`PROFIBUS_DL_SERVICE_ROLE_INITIATOR`). Otherwise use the packet “PROFIBUS DL CMD DLSAP ACTIVATE RESPONDER REQ” described in the next section.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DLSAP_ACTIVATE_REQ_Ttag {
  TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point to be activated */
  TLR_UINT8 bAcc;       /* access protection to specify if all stations or individual
station may access */
  TLR_UINT8 bParallelServices; /* Number of parallel Services that shall be activated the
same time */
  TLR_UINT8 bServiceActivate; /* services to be activated for that SAPIdx
SDA,SDN,SRD,MSRD,CS */
  TLR_UINT8 bRoleInService; /* service configuration INITIATOR, RESPONDER, BOTH */
  TLR_UINT8 bMaxDLSDULenReqLow; /* DLSDU length list */
  TLR_UINT8 bMaxDLSDULenReqHigh;
  TLR_UINT8 bMaxDLSDULenIndCnfLow;
  TLR_UINT8 bMaxDLSDULenIndCnfHigh;
} PROFIBUS_DL_DLSAP_ACTIVATE_REQ_T;

typedef struct  PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_DLSAP_ACTIVATE_REQ_T tData;
} PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	9	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x120	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DLSAP_ACTIVATE_REQ_T			
bSrcSAPIdx	UINT8	0-63, 64=NIL SAP	Source Service Access Point to be activated
bAcc	UINT8	0-126,255	Access protection to specify if all stations or individual station may access
bParallelServices	UINT8	1-4	Number of parallel Services that shall be activated the same time
bServiceActivate	UINT8	0-31	Services to be activated for that SAPIdx SDA,SDN,SRD,MSRD,CS
bRoleInService	UINT8	1-3	Service configuration INITIATOR, RESPONDER, BOTH
bMaxDLSDULenReqLow	UINT8	1-246	Maximum length of DLSDU list for requests with low priority
bMaxDLSDULenReqHigh	UINT8	1-246	Maximum length of DLSDU list for requests with high priority
bMaxDLSDULenIndCnfLow	UINT8	1-246	Maximum length of DLSDU list for indications and confirmations with low priority
bMaxDLSDULenIndCnfHigh	UINT8	1-246	Maximum length of DLSDU list for indications and confirmations with high priority

Table 165: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ - Activate a SAP for Request

Additional explanations:

1. The parameter bSrcSAPIdx specifies the source service access point of the received SRD frame. The allowed range for this parameter extends from 0 to 62. The value 64 is also allowed. It represents the NIL SAP. A value bSrcSAPIdx of 63 is not permitted in this command because it represents the global access.
2. The parameter bAcc is used for specifying the access protection for responder functions. It has the allowed values ALL = 255 or 0 to 126 and specifies whether all remote stations (ALL) or only an individual remote station identified by its station number (0 to 126) may have access to the LSAP. This parameter is only valid for responder functions, i.e. bRoleInService = RESPONDER, BOTH. If the access value is ALL, the device automatically sets the buffer length to the maximum value 244 and the services to all supported and the role in service to both independent what the rest command message parameter are.
3. The parameter bParallelServices specifies the upper limit for the number of parallel services that can be activated at the same time. Up to 4 parallel services are possible.
4. The parameter bServiceActivate contains the FDL service that shall be activated for this service according to the following table:

bServiceActivate parameter

Name	Value	Service
PROFIBUS_DL_SERVICE_SDA	0x01	SDA Service.
PROFIBUS_DL_SERVICE_SDN	0x02	SDN Service.
PROFIBUS_DL_SERVICE_SRD	0x04	SRD Service
PROFIBUS_DL_SERVICE_MSRD	0x08	MSRD Service
PROFIBUS_DL_SERVICE_CS	0x10	CS Service

Table 166: Values of bServiceActivate Parameter and the according Service

5. The parameter bRoleInService specifies the configuration for the service to be activated. The following values are specified:

bRoleInService parameter

Name	Value	Operation
PROFIBUS_DL_SERVICE_ROLE_INITIATOR	1	The device initiates the respective service exclusively.
PROFIBUS_DL_SERVICE_ROLE_RESPONDER	2	The device responds to the service exclusively. Not allowed for SRD.
PROFIBUS_DL_SERVICE_ROLE_BOTH	3	The device initiates and responds to the service. This option is not permissible for SRD.

Table 167: Values of bRoleInService Parameter and the according Operation

6. The parameters bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh specify the maximum length of the SDU for transparent data transfer depending on

   - whether low (bMaxDLSDULenXXXLow) or high priority (bMaxDLSDULenXXXHigh) is intended to be applied.
   - whether data are sent (in case of a request, bMaxDLSDULenReqYYY) or received (in case of either a confirmation or an indication, bMaxDLSDULenIndCnfYYY)

Additional rules apply for all of these parameters whether they must be equal to 0 or must not be equal to 0 depending on :

- the chosen service
- the role in service (Initiator, responder or both))

These rules can be expressed as tables. For each parameter and role in service intended to use in conjunction with this parameter there are a few allowed combinations whether to set the parameter to a zero- or non-zero-value which are represented by columns. If for a parameter the value 0 can be found it should be set to 0 otherwise (value ‘x’) it should be set to a non-zero value not exceeding 242:

bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh parameters for services SDA and SDN

Services SDA and SDN															
Name	**Role in service**														
	Initiator			**Responder**			**Both**								
bMaxDLSDULenReqLow	x	0	x	0	0	0	x	0	x	0	x	0	x	x	x
bMaxDLSDULenReqHigh	0	x	x	0	0	0	0	x	0	x	0	x	x	x	x
bMaxDLSDULenIndCnfLow	0	0	0	x	0	x	x	0	0	x	x	x	x	0	x
bMaxDLSDULenIndCnfHigh	0	0	0	0	x	x	0	x	x	0	x	x	0	x	x

Table 168: Allowed combinations of bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh parameters for services SDA and SDN

bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh parameters for services SRD and MSRD

Services SRD and MSRD												
Name	**Role in service**											
	Initiator											
bMaxDLSDULenReqLow	0	0	0	x	0	0	x	x	x	x	0	x
bMaxDLSDULenReqHigh	0	0	0	0	x	x	0	x	x	0	x	x
bMaxDLSDULenIndCnfLow	x	0	x	x	0	x	0	x	0	x	x	x
bMaxDLSDULenIndCnfHigh	0	x	x	0	x	0	x	0	x	x	x	x

Table 169: Allowed combinations of bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh parameters for services SRD and MSRD

For the roles in service *Responder* and *Both* no values are given as this packet is not applicable for the reply services SRD and MSRD, there is a special packet, please see the next section

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DLSAP_ACTIVATE_CNF_Ttag {
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point to be activated */
    TLR_UINT8 bAcc;       /* access protection to specify if all stations or individual
station may access */
  } PROFIBUS_DL_DLSAP_ACTIVATE_CNF_T;

  /* Confirmation-Packet for SAP activation */
  typedef struct  PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DLSAP_ACTIVATE_CNF_T tData;    /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	2	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x121	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_DLSAP_ACTIVATE_CNF_T			
bSrcSAPIdx	UINT8	0-63, 64	Source Service Access Point to be activated
bAcc	UINT8	0-126, 255	Access protection to specify if all stations or individual station may access

Table 170: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_CNF - Confirmation of Activate a SAP for Request

6.3.11 PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ/CNF - Activate an SAP for Responder Functionality

This packet allows configuring a local service access point for an SRD or MSRD service in a responder role. As outlined in section *Management of Services Access Points* on page 46, this special case requires a separate packet, namely this one.

Have in mind that this packet is only appropriate if the intended role in service (represented by the value of the `bRoleInService` parameter) is not “Initiator” (`PROFIBUS_DL_SERVICE_ROLE_INITIATOR`). Otherwise use the packet *PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request* described in the previous section.

For a more precise description of service access points refer to the section *Management of Services Access Points* on page 46.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_REQ_Ttag {
  TLR_UINT8 bSrcSAPIdx;         /* Source Service Access Point to be activated */
  TLR_UINT8 bAcc;               /* access protection to specify if all
                                   stations or individual station may access */
  TLR_UINT8 bParallelServices;  /* Number of parallel Services that shall be
                                   activated the same time */
  TLR_UINT8 bMaxDLSDULenReqLow; /* DLSDU length list */
  TLR_UINT8 bMaxDLSDULenReqHigh;
  TLR_UINT8 bMaxDLSDULenIndLow;
  TLR_UINT8 bMaxDLSDULenIndHigh;
  TLR_UINT8 bIndicationMode;
  TLR_BOOLEAN8 fPublisherEnabled;
} PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_REQ_T;

typedef struct  PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_RESPONDER_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_REQ_T tData;
} PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_RESPONDER_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_RESPONDER_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	9	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x122	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_REQ_T			
bSrcSAPIdx	UINT8	0-63, 64 (=NIL SAP)	Source Service Access Point to be activated
bAcc	UINT8	0-126,255	access protection to specify if all stations or individual station may access
bParallelServices	UINT8	1-4	Number of parallel Services that shall be activated the same time
bMaxDLSDULenReqLow	UINT8	1-246	DLSDU length list
bMaxDLSDULenReqHigh	UINT8	1-246	DLSDU length list
bMaxDLSDULenIndLow	UINT8	1-246	DLSDU length list
bMaxDLSDULenIndHigh	UINT8	1-246	DLSDU length list
bIndicationMode	UINT8	0-2	Indication mode
fPublisherEnabled	BOOLEAN	0,1	Enable or disable publisher functionality

Table 171: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ - Activate a SAP for Responder Functionality

Additional explanations:

1. The parameter bSrcSAPIdx specifies the local LSAP for the data area and the user that receives the indication primitive if a SRD is received on this "DSAP". The value can have a range from 0 to 63 and NIL is represented by 64.
2. The parameter bAcc with the values ALL = 255 or 0 to 126 is used for access protection and specifies whether all remote stations (all) or only an individual remote station (0 to 126) may have access to the LSAP. This parameter is only valid for responder functions, i.e. bRoleInService = RESPONDER or BOTH.
3. The parameter bParallelServices specifies the upper limit for the number of parallel services that can be activated at the same time. Up to 4 parallel services are possible.
4. The parameters bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndLow and bMaxDLSDULenIndHigh specify the maximum length of the SDU for transparent data transfer depending on
   - whether low (bMaxDLSDULenXXXLow) or high priority (bMaxDLSDULenXXXHigh) is intended to be applied.
   - whether data are sent (in case of a request, bMaxDLSDULenReqYYY) or received (in case of either a confirmation or an indication, bMaxDLSDULenIndYYY)
   - Additional rules apply for all of these parameters whether they must be equal to 0 or must not be equal to 0. These rules can be expressed as tables. For each parameter and role in service intended to use in conjunction with this parameter there are a few allowed combinations whether to set the parameter to a zero- or non-zero-value which are represented by columns. If for a parameter the value 0 can be found it should be set to 0 otherwise (value ‘x’) it should be set to a non-zero value not exceeding 242:

bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndLow and bMaxDLSDULenIndHigh parameters for services SRD and MSRD in ‘Responder’ role

Services SRD and MSRD												
Name	**Role in service**											
	Responder or both											
bMaxDLSDULenReqLow	x	0	x	x	0	0	x	x	x	x	0	x
bMaxDLSDULenReqHigh	0	x	x	0	x	x	0	x	x	0	x	x
bMaxDLSDULenIndLow	0	0	0	x	0	x	0	x	0	x	x	x
bMaxDLSDULenIndHigh	0	0	0	0	x	0	x	0	x	x	x	x

Table 172: Allowed combinations of length parameters for service SRD with role in service ‘Responder’

5. The parameter `bIndicationMode` specifies when to create an indication at the receiver of the packet. The following table shows the allowed values of the parameter `bUpdateStatus` and their meanings:

Name	Value	Meaning
PROFIBUS_DL_INDICATION_MODE_ALL	0x00	An indication is generated always, even in case of zero data length.
PROFIBUS_DL_INDICATION_MODE_DATA	0x01	An indication is generated if data transfer really happens, i.e. only in case of non-zero data length of at least either the request or the reply data.
PROFIBUS_DL_INDICATION_MODE_UNCHANGED	0x02	This option is used only for changing the bAcc parameter. It is used in the following manner: The bAcc parameter will be changed according to its setting within the packet. All other parameters will completely remain unchanged.

Table 173: Allowed Values and their Meanings for Variable `bUpdateStatus`

The parameter `fPublisherEnabled` specifies the reply update mode, i.e. whether or not the service using the RSAP may act as publisher or not. The possible values in this context are TRUE(1) or FALSE.(0). However, this parameter is currently not supported.

Packet Structure Reference

```
  typedef struct PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_CNF_Ttag {
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point to be activated */
    TLR_UINT8 bAcc;       /* access protection to specify if all stations or individual
station may access */
  } PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_CNF_T;

  /* Request-Packet for SAP responder activation */
  typedef struct  PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_RESPONDER_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_CNF_T tData;    /* Packet Data Unit */
  } PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_RESPONDER_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DLSAP_ACTIVATE_RESPONDER_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	2	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 “Packets for Configuration during running system”.
ulCmd	UINT32	0x123	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_DLSAP_ACTIVATE_RESPONDER_CNF_T			
bSrcSAPIdx	UINT8	0-63. 64 = NIL SAP	Source Service Access Point to be activated
bAcc	UINT8	0-126, 255	access protection to specify if all stations or individual station may access

Table 174: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_CNF – Confirmation of Activate a SAP for Responder Functionality

6.3.12 PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request

This service is used to deactivate a local service access point which has previously been defined either with the

- PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request packet (page 228) or with the
- PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request packet (page 238).

Only the numerical value assigned with the SAP or the RSAP to be deactivated needs to be specified.

Packet Structure Reference

```
/* Request-Packet for SAP deactivation */
  typedef struct PROFIBUS_DL_DLSAP_DEACTIVATE_REQ_Ttag {
    TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point to be activated */
  } PROFIBUS_DL_DLSAP_DEACTIVATE_REQ_T;

  typedef struct  PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_REQ_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_DLSAP_DEACTIVATE_REQ_T tData;
  } PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	1	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x128	PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DLSAP_DEACTIVATE_REQ_T			
bSrcSAPIdx	UINT8	0-63, 64	Source Service Access Point to be deactivated

Table 175: PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ - Deactivate an SAP for Request

Packet Structure Reference

```
typedef struct PROFIBUS_DL_DLSAP_DEACTIVATE_CNF_Ttag {
  TLR_UINT8 bSrcSAPIdx; /* Source Service Access Point to be activated */
} PROFIBUS_DL_DLSAP_DEACTIVATE_CNF_T;

/* Confirmation-Packet for SAP deactivation */
typedef struct  PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_DLSAP_DEACTIVATE_CNF_T tData;    /* Packet Data Unit */
} PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	1	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x129	PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_DLSAP_DEACTIVATE_CNF_T			
bSrcSAPIdx	UINT8	0-63, 64	Source Service Access Point which has been deactivated in case of successful execution

Table 176: PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_CNF – Confirmation of Deactivate an SAP for Request

6.3.13 PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set

This packet can be applied to load a bus parameter set to the communication channel. A detailed description of all bus parameters is given in section 4.2 "*Detailed Description of Bus and Master Parameters*" of this document.

A data structure containing the configured bus and master parameter set is returned along with the confirmation packet.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Note: This packet belongs to layer 2 (data link layer) of the OSI-layer model for networks. There is also a packet with similar functionality which allows setting bus parameters and is located at layer 7 (the application layer): the packet "*PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register*" described in section 6.1.1 of this document.

Note: The former parameter `bAlarm_Max` is no longer supported. It has been replaced by the new parameter `bMasterSetting`.

Packet Structure Reference

```
typedef struct  PROFIBUS_DL_PACKET_SET_BUS_PARAMETER_SET_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_BUS_PARAMETER_SET_T tData;
} PROFIBUS_DL_PACKET_SET_BUS_PARAMETER_SET_REQ_T;

typedef struct PROFIBUS_DL_BUS_PARAMETER_SET_Ttag {
  TLR_UINT16 usBus_Para_Len; /* Contains the length of the Bus_Para inclusive the field
Bus_Para_Len itself */
  TLR_UINT8  bDL_Add; /* Contains the own address of the PROFIBUS Device */
  TLR_UINT8  bData_rate; /* Contains the Transmission speed */
  TLR_UINT16 usTSL; /* slot-time */
  TLR_UINT16 usMin_TSDR; /* min. station delay responder */
  TLR_UINT16 usMax_TSDR; /* max. station delay responder */
  TLR_UINT8  bTQUI; /* quiet time */
  TLR_UINT8  bTSET; /* setup time */
  TLR_UINT32 ulTTR; /* target rotation time */
  TLR_UINT8  bG; /* Gap update factor */
  TLR_UINT8  bHSA; /* Highest station address */
  TLR_UINT8  bMax_Retry_Limit; /* retries if error occurs */
  TLR_UINT8  bBp_Flag;
  TLR_UINT16 usMin_Slave_Interval; /* Minimum Slave Interval Time */
  TLR_UINT16 usPoll_Timeout; /* Class2 Poll timeout */
  TLR_UINT16 usData_Control_Time; /* Data Control Time */
  /*TLR_UINT8  bAlarm_Max; *//* Maximum Alarms */
  TLR_UINT8  bMasterSetting; /* 0 == big Endian, 1 == little Endian)*/
  TLR_UINT8  bMax_User_Global_Control; /* Maximum allowed parallel active USER Global
Control Commands */
  TLR_UINT8  abReserved[4]; /* 4 reserved Octets */
  TLR_UINT16 usMaster_User_Data_Len; /* Contains the length of the USER Master data */
  TLR_UINT8  abMaster_Class2_Name[32]; /* Name of the Master */
  TLR_UINT8  abMaster_User_Data[32]; /* USER specific Parameter data */
  TLR_UINT32 ulTCL; /* Isochronous cycle time */
  TLR_UINT8  bMax_TSH; /* Maximum Shift Time */
} PROFIBUS_DL_BUS_PARAMETER_SET_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_SET_BUS_PARAMETER_SET_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	>=19	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task*
ulCmd	UINT32	0x126	PROFIBUS_DL_CMD_SET_BUS_PARAMETER_SET_REQ - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_BUS_PARAMETER_SET_T			
usBus_Para_Len	UINT16		Contains the length of the Bus_Para including the field usBus_Para_Len itself sizeof(PROFIBUS_DL_BUS_PARAMETER_SET_T) **Note**: This parameter is not evaluated by the PROFIBUS DP Master firmware. It is only present for compatibility reasons.
bDL_Add	UINT8	0-126	Contains the own address of the PROFIBUS Device
bData_rate	UINT8		Contains the Transmission speed
usTSL	UINT16		Slot-time
usMin_TSDR	UINT16		Min. station delay responder
usMax_TSDR	UINT16		Max. station delay responder
bTQUI	UINT8		Quiet time
bTSET	UINT8		Setup time
ulTTR	UINT32		Target rotation time
bG	UINT8	1-100	Gap update factor
bHSA	UINT8	0-126	Highest station address
bMax_Retry_Limit	UINT8	1-15	Retries if error occurs
bBp_Flag	UINT8		Bp flag
usMin_Slave_Interval	UINT16		Minimum Slave Interval Time
usPoll_Timeout	UINT16		Class2 Poll timeout
usData_Control_Time	UINT16		Data Control Time
bMasterSetting	UINT8	0,1	Master Setting 0: Big Endian/Motorola format 1: Little Endian/Intel format
bMax_User_Global_Control	UINT8		Maximum allowed parallel active USER Global Control Commands
abReserved	UINT8[4]		4 reserved Octets
usMaster_User_Data_Len	UINT16		Contains the length of the USER Master data
abMaster_Class2_Name[32]	UINT8[32]		Name of the Master

abMaster_User_Data[32];	UINT8[32]		USER specific Parameter data
ulTCL	UINT32		Isochronous cycle time
bMax_TSH	UINT8		Maximum Shift Time

Table 177: PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ - Load the Bus Parameter Set

Packet Structure Reference

```
  /* Confirmation-Packet for the setting the DL parameter */
  typedef struct  PROFIBUS_DL_PACKET_SET_BUS_PARAMETER_SET_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_DL_BUS_PARAMETER_SET_T tData;
  } PROFIBUS_DL_PACKET_SET_BUS_PARAMETER_SET_CNF_T;

typedef struct PROFIBUS_DL_BUS_PARAMETER_SET_Ttag {
  TLR_UINT16 usBus_Para_Len; /* Contains the length of the Bus_Para inclusive the field
Bus_Para_Len itself */
  TLR_UINT8  bDL_Add; /* Contains the own address of the PROFIBUS Device */
  TLR_UINT8  bData_rate; /* Contains the Transmission speed */
  TLR_UINT16 usTSL; /* slot-time */
  TLR_UINT16 usMin_TSDR; /* min. station delay responder */
  TLR_UINT16 usMax_TSDR; /* max. station delay responder */
  TLR_UINT8  bTQUI; /* quite time */
  TLR_UINT8  bTSET; /* setup time */
  TLR_UINT32 ulTTR; /* target rotation time */
  TLR_UINT8  bG; /* Gap update factor */
  TLR_UINT8  bHSA; /* Highest station address */
  TLR_UINT8  bMax_Retry_Limit; /* retries if error occurs */
  TLR_UINT8  bBp_Flag;
  TLR_UINT16 usMin_Slave_Interval; /* Minimum Slave Interval Time */
  TLR_UINT16 usPoll_Timeout; /* Class2 Poll timeout */
  TLR_UINT16 usData_Control_Time; /* Data Control Time */
  TLR_UINT8  bAlarm_Max; /* Maximum Alarms */
  TLR_UINT8  bMax_User_Global_Control; /* Maximum allowed parallel active USER Global
Control Commands */
  TLR_UINT8  abReserved[4]; /* 4 reserved Octets */
  TLR_UINT16 usMaster_User_Data_Len; /* Contains the length of the USER Master data */
  TLR_UINT8  abMaster_Class2_Name[32]; /* Name of the Master */
  TLR_UINT8  abMaster_User_Data[32]; /* USER specific Parameter data */
  TLR_UINT32 ulTCL; /* Isochronous cycle time */
  TLR_UINT8  bMax_TSH; /* Maximum Shift Time */
} PROFIBUS_DL_BUS_PARAMETER_SET_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_SET_VALUE_BUS_PARAMETER_SET_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	103	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter Packets for Configuration during running system.
ulCmd	UINT32	0x127	PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_CNF - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change

structure PROFIBUS_DL_BUS_PARAMETER_SET_T			
usBus_Para_Len	UINT16		Contains the length of the Bus_Para including the field `usBus_Para_Len` itself
bDL_Add	UINT8		Contains the own address of the PROFIBUS Device
bData_rate	UINT8		Contains the Transmission speed
usTSL	UINT16		slot-time
usMin_TSDR	UINT16		min. station delay responder
usMax_TSDR	UINT16		Max. station delay responder
bTQUI	UINT8		quiet time
bTSET	UINT8		setup time
ulTTR	UINT32		target rotation time
bG	UINT8		Gap update factor
bHSA	UINT8		Highest station address
bMax_Retry_Limit	UINT8		retries if error occurs
bBp_Flag	UINT8		
usMin_Slave_Interval	UINT16		Minimum Slave Interval Time
usPoll_Timeout	UINT16		Class2 Poll timeout
usData_Control_Time	UINT16		Data Control Time
bAlarm_Max	UINT8		Maximum Alarms
bMax_User_Global_Control	UINT8		Maximum allowed parallel active USER Global Control Commands
abReserved	UINT8[4]		4 reserved Octets
usMaster_User_Data_Len	UINT16		Contains the length of the USER Master data
abMaster_Class2_Name[32]	UINT8[32]		Name of the Master
abMaster_User_Data[32];	UINT8[32]		USER specific Parameter data
ulTCL	UINT32		Isochronous cycle time
bMax_TSH	UINT8		Maximum Shift Time

Table 178: PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_CNF – Confirmation of Loading the Bus Parameter Set

6.3.14 PROFIBUS_DL_CMD_SET_VALUE_REQ/CNF - Set Value Service

This service sets a specific value within the variable set of the DL Layer.

The following system variables are available and supported by this packet:

1. Bus parameter minT_{SDR}

   Indicates the smallest station delay response time. See section *Min TSDR* for detailed information. Allowed values range from 1 to 65535 (= 2^{16}-1).

2. Own Address DL_ADDR

   Indicates the own address of the PROFIBUS DP master that can be changed only by this method. Allowed values range from 0 to 126.

Note: Use this packet only when working with linkable object modules. It has not been designed for usage in the context of loadable firmware.

Packet Structure Reference

```
#define PROFIBUS_DL_SET_VALUE_MINTDSR (0)
#define PROFIBUS_DL_SET_VALUE_DLADDR (1)

typedef struct PROFIBUS_DL_SET_VALUE_REQ_Ttag
{
  TLR_UINT8 bVlu; /* Value to be set */
  union {
    TLR_UINT16 usMin_TSDR;
    TLR_UINT8 bDl_Add;
  } un;
}
PROFIBUS_DL_SET_VALUE_REQ_T;

/* Request-Packet for setting a DL value */
typedef struct  PROFIBUS_DL_PACKET_SET_VALUE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_SET_VALUE_REQ_T tData;
}
PROFIBUS_DL_PACKET_SET_VALUE_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_SET_VALUE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	PB_DL_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	 3 2	Packet data length in bytes depends on bVlu only if bVlu has been set to 0 only if bVlu has been set to 1
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x12A	PROFIBUS_DL_CMD_SET_VALUE_REQ - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_SET_VALUE_REQ_T			
bVlu	UINT8	0,1	Value to be set: 0: MINTDSR 1: DLADDR
un	UINT16 UINT8	1…65535 0…126	minT_{SDR} (only if bVlu has been set to 0) bDl_Add (only if bVlu has been set to 1)

Table 179: PROFIBUS_DL_CMD_SET_VALUE_REQ - Set Value Service

Packet Structure Reference

```
/* Confirmation-Packet for setting a DL value */
  typedef struct  PROFIBUS_DL_PACKET_SET_VALUE_CNF_Ttag {
    TLR_PACKET_HEADER_T tHead;
  } PROFIBUS_DL_PACKET_SET_VALUE_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_SET_VALUE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, unchanged
ulSrcId	UINT32	ulDL0Id	Source end point identifier, unchanged
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		See chapter 6.5 Packets for Configuration during running system
ulCmd	UINT32	0x12B	`PROFIBUS_DL_CMD_SET_VALUE_CNF` - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 180: PROFIBUS_DL_CMD_SET_VALUE_CNF – Confirmation of Set Value Service

6.3.15 PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ/CNF – Obtain FDL Status

This packet is used for obtaining the FDL status of a specific PROFIBUS DP slave. According to the PROFIBUS DP specification, the FDL status is defined as “the token ring status of an FDL entity conveyed in the FC field of the reply frames”. This also exactly represents the entry of this specific slave in the life list.

Packet Structure Reference

```
typedef struct PROFIBUS_DL_SEND_FDL_STATUS_REQ_Ttag
{
  TLR_UINT8 bDstAddr; /* Value to be set */
}
PROFIBUS_DL_SEND_FDL_STATUS_REQ_T;

typedef struct  PROFIBUS_DL_PACKET_SEND_FDL_STATUS_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_SEND_FDL_STATUS_REQ_T tData;
}
PROFIBUS_DL_PACKET_SEND_FDL_STATUS_REQ_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_SEND_FDL_STATUS_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ PB_DL_QUE	Destination Queue-Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	ulDL0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task*
ulCmd	UINT32	0x132	PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_DL_SEND_FDL_STATUS_REQ_T			
bDstAddr	UINT8		Slave address whose status is to be requested

Table 181: PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ – Obtain FDL Status Request

Packet Structure Reference

```
typedef struct PROFIBUS_DL_SEND_FDL_STATUS_CNF_Ttag
{
  TLR_UINT8 bStatus;
}
PROFIBUS_DL_SEND_FDL_STATUS_CNF_T;

typedef struct  PROFIBUS_DL_PACKET_SEND_FDL_STATUS_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_SEND_FDL_STATUS_CNF_T tData;     /* Packet Data Unit */
}
PROFIBUS_DL_PACKET_SEND_FDL_STATUS_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_SEND_FDL_STATUS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	ulAPM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task.*
ulCmd	UINT32	0x133	PROFIBUS_DL_CMD_SEND_FDL_STATUS_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_DL_SEND_FDL_STATUS_CNF_T			
bStatus	UINT8		Requested status of addressed slave

Table 182: PROFIBUS_DL_CMD_SEND_FDL_STATUS_CNF – Confirmation of Obtain FDL Status Request

6.3.16 PROFIBUS_DL_CMD_GET_LMS_REQ/CNF - Get List of active Stations

This packet allows accessing the “list of master stations” (LMS) in the logical ring also often called “list of active stations” (LAS). This list is represented by an array of 128 elements, of which, however, at most 127 are used.

Each array element (i.e. entry within the list) represents a station address within the PROFIBUS network.

- If there is either no active station present at this station address or if there is a PROFIBUS Slave at this station address, the corresponding array element will contain the value 255 (0xFF).
- If there is a PROFIBUS DP Master at this station address, the corresponding array element will contain the FDL address of the PROFIBUS Master to which this Master will pass the token next.

Note: A valid station address is defined in PROFIBUS to be a value in the range between 0 and 126 (including both limiting values).

The request packet does not have any parameters. The confirmation packet has one parameter, the returned array `abLms[128]` containing the “list of master stations”.

Packet Structure Reference

```
/* Get List of active stations */
typedef struct  PROFIBUS_DL_PACKET_GET_LMS_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}
PROFIBUS_DL_PACKET_GET_LMS_REQ_T;

#define PROFIBUS_DL_GET_LMS_REQ_SIZE (0)
```

Packet Description

structure PROFIBUS_DL_PACKET_GET_LMS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task*
ulCmd	UINT32	0x0134	`PROFIBUS_DL_CMD_GET_LMS_REQ` - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	*UINT32*	*x*	*Routing, do not change*

Table 183: PROFIBUS_DL_CMD_GET_LMS_REQ - Get List of active Stations

Packet Structure Reference

```
/* Confirmation Packet for get LMS */
typedef struct PROFIBUS_DL_GET_LMS_CNF_Ttag
{
  TLR_UINT8 abLms[128];
}
PROFIBUS_DL_GET_LMS_CNF_T;

typedef struct  PROFIBUS_DL_PACKET_GET_LMS_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_GET_LMS_CNF_T tData;     /* Packet Data Unit */
} PROFIBUS_DL_PACKET_GET_LMS_CNF_T;

#define PROFIBUS_DL_GET_LMS_CNF_SIZE (sizeof(PROFIBUS_DL_GET_LMS_CNF_T))
```

Packet Description

structure PROFIBUS_DL_PACKET_GET_LMS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	128	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 “Error Codes of the DL-Task”*
ulCmd	UINT32	0x0135	PROFIBUS_DL_CMD_GET_LMS_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change
structure PROFIBUS_DL_GET_LMS_CNF_T			
abLms[128]	UINT8[]	0…126, 255 (for each array element)	List of active Stations

Table 184: PROFIBUS_DL_CMD_GET_LMS_CNF - – Confirmation of Get List of active Stations Request

6.3.17 PROFIBUS_DL_CMD_REGISTER_LMS_REQ/CNF - Register an Application to receive LMS Change Indications

This packet allows registering in order to be informed about current changes in the list of master stations (LMS) also often denominated as list of active stations (LAS). Every time a change within this list occurs, the PROFIBUS DL task of the protocol stack will receive an indication packet (PROFIBUS_DL_CMD_LMS_CHANGED_IND - Indication for Change in LMS, see next subsection) indicating that a change has occurred and where this change has occurred.

Each array element (i.e. entry within the list) represents a station address within the PROFIBUS network.

For more information about the list, see the preceding subsection beginning on page 249 of this document (PROFIBUS_DL_CMD_GET_LMS_REQ/CNF - Get List of active Stations).

The confirmation packet does not have any parameters.

Packet Structure Reference

```
/* Register an application to receive LMS change indications*/
typedef struct PROFIBUS_DL_REGISTER_LMS_REQ_Ttag
{
  TLR_BOOLEAN8 bEnable; /* Value to be set */
}
PROFIBUS_DL_REGISTER_LMS_REQ_T;

typedef struct  PROFIBUS_DL_PACKET_REGISTER_LMS_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_REGISTER_LMS_REQ_T tData;
}
PROFIBUS_DL_PACKET_REGISTER_LMS_REQ_T;

#define PROFIBUS_DL_REGISTER_LMS_REQ_SIZE (sizeof(PROFIBUS_DL_REGISTER_LMS_REQ_T))
```

Packet Description

structure PROFIBUS_DL_PACKET_REGISTER_LMS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task*
ulCmd	UINT32	0x0136	`PROFIBUS_DL_CMD_REGISTER_LMS_REQ` - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_REGISTER_LMS_REQ_T			
bEnable	BOOLEAN8	0,1	Value to be set 0: Indications are switched off 1: Indications are switched on

Table 185: PROFIBUS_DL_CMD_REGISTER_LMS_REQ - Register an Application to receive LMS Change Indications

Packet Structure Reference

```
/* Register an application to receive LMS change indications*/
typedef struct  PROFIBUS_DL_PACKET_REGISTER_LMS_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_REGISTER_LMS_CNF_T;

#define PROFIBUS_DL_REGISTER_LMS_CNF_SIZE (0)
```

Packet Description

structure PROFIBUS_DL_PACKET_REGISTER_LMS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulAPM0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 Error Codes of the DL-Task*
ulCmd	UINT32	0x0137	PROFIBUS_DL_CMD_REGISTER_LMS_CNF- Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	×	Routing, do not change

Table 186: PROFIBUS_DL_CMD_REGISTER_LMS_CNF – Confirmation of Register an Application to receive LMS Change Indications Request

6.3.18 PROFIBUS_DL_CMD_LMS_CHANGED_IND - Indication for Change in LMS

Every time a change within the list of master stations (LMS) occurs, the PROFIBUS DL task of the protocol stack will receive this indication packet indicating that a change has occurred and where this change has occurred.

Each array element (i.e. entry within the list) contains the FDL address of a currently active PROFIBUS Master within the PROFIBUS network.

For more information about the list, refer to subsection PROFIBUS_DL_CMD_GET_LMS_REQ/CNF - Get List of active Stations beginning on page 249 of this document.

Receiving indications requires registration via the packet PROFIBUS_DL_CMD_REGISTER_LMS_REQ/CNF - Register an Application to receive LMS Change Indications. For more information about the list, see the preceding subsection beginning on page 252 of this document.

The indication packet has one parameter, the array `abLms[128]` containing the “list of master stations”.

Packet Structure Reference

```
/* Indication for changed LMS */
typedef struct PROFIBUS_DL_LMS_CHANGED_IND_Ttag
{
  TLR_UINT8 abLms[128];
}
PROFIBUS_DL_LMS_CHANGED_IND_T;

#define PROFIBUS_DL_LMS_CHANGED_IND_SIZE (sizeof(PROFIBUS_DL_LMS_CHANGED_IND_T))

/* Indication Packet for acknowledged connectionless data transfer */
typedef struct  PROFIBUS_DL_PACKET_LMS_CHANGED_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_LMS_CHANGED_IND_T tData;
}
PROFIBUS_DL_PACKET_LMS_CHANGED_IND_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_LMS_CHANGED_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32	ulDL0Id	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	128	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet identification as unique number generated by the Source process of the packet
ulSta	UINT32		*See section 8.4 "Error Codes of the DL-Task"*
ulCmd	UINT32	0x0138	`PROFIBUS_DL_CMD_LMS_CHANGED_IND` - Command
ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_DL_DATA_REPLY_REQ_T			
abLms[128]	UINT8[]	0…126,255 (for each array element)	List of active Stations

Table 187: PROFIBUS_DL_CMD_LMS_CHANGED_IND - Indication for acknowledged connectionless Data Transfer

6.3.19 PROFIBUS_DL_CMD_GET_LL_REQ/CNF – Get the Life List (List of Active or Passive Stations)

This packet allows to retrieve the life list, i.e. a list indicating which stations are currently active and which are passive.

Each array element (i.e. entry within the list) represents a station address within the PROFIBUS network.

Note: A valid station address is defined in PROFIBUS to be a value in the range between 0 and 126 (including both limiting values).

The single bytes in array `abLL[128]` in the response have the following meaning:

Value	Symbolic name	Meaning
0	PROFIBUS_DL_STATION_TYPE_NOT_EXISTANT	Station does not exist
1	PROFIBUS_DL_STATION_TYPE_PASSIVE	Passive station (i.e. PROFIBUS slave) at this address
2	PROFIBUS_DL_STATION_TYPE_READY_NO_TOKEN	Active station not ready for token ring at this address
3	PROFIBUS_DL_STATION_TYPE_ACTIVE	Active station

Table 188: Possible Values of Single Bytes in Response

Packet Structure Reference

```
/* Request-Packet to get list of active station */
typedef struct  PROFIBUS_DL_PACKET_GET_LL_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
}
PROFIBUS_DL_PACKET_GET_LL_REQ_T;
#define PROFIBUS_DL_GET_LL_REQ_SIZE (0)
```

Packet Description

structure PROFIBUS_DL_PACKET_GET_LL_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	ulDL0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulAPMS0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.4 “Error Codes of the DL-Task”*
ulCmd	UINT32	0x0130	`PROFIBUS_DL_CMD_GET_LL_REQ` - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 189: PROFIBUS_DL_CMD_GET_LL_REQ - Get Life List Request

Packet Structure Reference

```
/* Confirmation-Packet for get list of active station */
typedef struct  PROFIBUS_DL_GET_LL_CNF_Ttag
{
  TLR_UINT8 abLL[128];
}
PROFIBUS_DL_GET_LL_CNF_T;

typedef struct  PROFIBUS_DL_PACKET_GET_LL_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_DL_GET_LL_CNF_T tData;     /* Packet Data Unit */
}
PROFIBUS_DL_PACKET_GET_LL_CNF_T;
```

Packet Description

structure PROFIBUS_DL_PACKET_GET_LL_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	ulAPM0Id	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	ulDL0Id	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	128	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.4 “Error Codes of the DL-Task”*
ulCmd	UINT32	0x0131	PROFIBUS_DL_CMD_GET_LL_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
structure PROFIBUS_DL_GET_LL_CNF_T			
abLL[128]	UINT8[]	0..3	Life List (see explanation above)

Table 190: PROFIBUS_DL_CMD_GET_LL_CNF - Confirmation to Get Life List Request

6.4 The APM-Task

The APM-Task only offers packets regarding redundancy function. Nevertheless you should be aware of its existence as it is very important for performing internal processes. These concern for instance such important things as:

- Diagnosis
- Routing of packets
- Database operations
- Data Exchange with Dual Port Memory.
- Error processing

To get the handle of the process queue of the APM -Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII queue name	Description
"APM_QUE"	Name of the APM-Task process queue

Table 191: FSPMM-Task process queue

The diagnostic and error messages which can be issued by the APM-Task are described in section 8.5 of this document.

In detail, the following functionality is provided by the APM -Task:

Overview over Packets of the APM-Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.4.1	PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ/CNF – Set Master to active/passive	0x3000/ 0x3001	261
6.4.2	PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master	0x3002	264

Table 192: Overview over the Packets of the APM-Task of the PROFIBUS DP Protocol Stack

6.4.1 PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ/CNF – Set Master to active/passive

This packet allows to change the behavior of the master between active and passive master. It also allows switching between the standard redundancy behavior (which has been defined by the PNO and excludes DPV1 slaves) and a special Hilscher mode allowing DPV1 slaves to participate.

For more information on this topic, please refer to section *Redundancy Functionality* on page 52.

Packet Structure Reference

```
#define PROFIBUS_APM_REDUNDANT_MASTER_ACTIVE          0x00
#define PROFIBUS_APM_REDUNDANT_MASTER_BACKUP          0x01

#define PROFIBUS_APM_REDUNDANT_MODE_PNO               0x00
#define PROFIBUS_APM_REDUNDANT_MODE_HILSCHER          0x02

#define PROFIBUS_APM_REDUNDANT_MONITOR_PRIMARY_ONLY   0x00
#define PROFIBUS_APM_REDUNDANT_MONITOR_BACKUP_PRIMARY 0x04

typedef struct  PROFIBUS_APM_REDUNDANT_MODE_REQ_Ttag
{
  TLR_UINT8 bRedundantAddr;     /* station address of redundant master */
  TLR_UINT8 bOption;            /* redundant options */
} PROFIBUS_APM_REDUNDANT_MODE_REQ_T;

#define PROFIBUS_APM_REDUNDANT_MODE_REQ_SIZE sizeof(PROFIBUS_APM_REDUNDANT_MODE_REQ_T)

typedef struct  PROFIBUS_APM_PACKET_REDUNDANT_MODE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_APM_REDUNDANT_MODE_REQ_T tData;    /* Packet Data Unit */
} PROFIBUS_APM_PACKET_REDUNDANT_MODE_REQ_T;
```

Packet Description

structure PROFIBUS_APM_PACKET_REDUNDANT_MODE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ APM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	0	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	Freely eligible	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*See section 8.5 Error Codes of the APM-Task*
ulCmd	UINT32	0x3000	PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_APM_REDUNDANT_MODE_REQ_T			
bRedundantAddr	UINT8	0...126 255	Address of the redundant master. Value 255 is for using the address of the active master minus 1 for bOption = 1. If the active master has address 0, then address 125 is used.
bOption	UINT8	Bit 0: 0 = active 1 = passive Bit 1: 0 = PNO Mode 1 = Hilscher Mode Bit 2: 0 = monitor primary only 1 = monitor backup and primary Bit 3-7: reserved	Bit mask for redundant options Bit 0 decides between active and passive master. Bit 1 decides between PNO and Hilscher mode of redundancy. Bit 2 decides monitoring of status of partner and indicating with PROFIBUS_APM_PACKET_REDUNDANT_MODE_IND

Table 193: PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ

Packet Structure Reference

```
#define PROFIBUS_APM_REDUNDANT_MODE_CNF_SIZE (0)

typedef struct  PROFIBUS_APM_PACKET_REDUNDANT_MODE_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} PROFIBUS_APM_PACKET_REDUNDANT_MODE_CNF_T;
```

Packet Description

structure PROFIBUS_APM_PACKET_REDUNDANT_MODE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32	Value of preceding request packet is returned	Destination end point identifier, unchanged
ulSrcId	UINT32	Value of preceding request packet is returned	Source end point identifier, unchanged
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section 8.5 Error Codes of the APM-Task*
ulCmd	UINT32	0x3001	PROFIBUS_APM_CMD_REDUNDANT_MODE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change

Table 194: PROFIBUS_APM_CMD_REDUNDANT_MODE_CNF

6.4.2 PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master

This packet indicates when the status of the primary master has changed at the local LAS. If the feature to monitor the backup master is enabled, this packet is also indicating change of status of the backup master. For more information on this topic, please refer to section *Redundancy Functionality* on page 52.

Packet Structure Reference

```
typedef struct  PROFIBUS_APM_REDUNDANT_MODE_IND_Ttag
{
  TLR_BOOLEAN8 fActiveMasterExist;
  TLR_BOOLEAN8 fBackupMasterExist;
} PROFIBUS_APM_REDUNDANT_MODE_IND_T;

#define PROFIBUS_APM_REDUNDANT_MODE_IND_SIZE (sizeof(PROFIBUS_APM_REDUNDANT_MODE_IND_T))

typedef struct  PROFIBUS_APM_PACKET_REDUNDANT_MODE_IND_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  PROFIBUS_APM_REDUNDANT_MODE_IND_T tData;
} PROFIBUS_APM_PACKET_REDUNDANT_MODE_IND_T;
```

Packet Description

structure PROFIBUS_APM_PACKET_REDUNDANT_MODE_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ APM_QUE	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32	Value of preceding request packet is returned	Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0	Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	2	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*See section 8.5 Error Codes of the APM-Task*
ulCmd	UINT32	0x3002	PROFIBUS_APM_CMD_REDUNDANT_MODE_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure PROFIBUS_APM_REDUNDANT_MODE_IND_T			
fActiveMasterExist	UINT8	TRUE FALSE	Status of primary master
fBackupMasterExist	UINT8	TRUE FALSE	Status of backup master

Table 195: PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master

6.5 Packets for Configuration during running system

The following packets are defined within rcX, but may be of special interest when working with configuration “on the run”:

In detail, the following functionality is described within this section:

Overview over Packets required for Configuration “on the run”			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.5.1	RCX_VERIFY_DATABASE_REQ/CNF – Verify the new Configuration Database	0x2F82/ 0x2F83	266
6.5.2	RCX_ACTIVATE_DATABASE_REQ/CNF – Activate the new Configuration Database	0x2F84/ 0x2F85	269

Table 196: Overview over Packets required for Configuration “on the run”

6.5.1 RCX_VERIFY_DATABASE_REQ/CNF – Verify the new Configuration Database

This packet informs the PROFIBUS DP Master, that a new configuration database has been downloaded and needs to be verified.

Packet Structure Reference

```
typedef struct  RCX_VERIFY_DATABASE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;                    /* packet header */
} RCX_VERIFY_DATABASE_REQ_T;

#define RCX_VERIFY_DATABASE_REQ_SIZE sizeof(RCX_VERIFY_DATABASE_REQ_T)
```

Packet Description

structure RCX_VERIFY_DATABASE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ `rcX queue handle`	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32		Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		
ulCmd	UINT32	0x2F82	`RCX_VERIFY_DATABASE_REQ` - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 197: RCX_VERIFY_DATABASE_REQ – Verify the new Configuration Database

Packet Structure Reference

```
typedef struct RCX_VERIFY_SLAVE_DATABASE_LIST_Ttag
{
   TLR_UINT32   ulLen;
   TLR_UINT8    ausData[16];
} RCX_VERIFY_SLAVE_DATABASE_LIST_T;


typedef struct RCX_VERIFY_DATABASE_CNF_DATA_Ttag
{
   RCX_VERIFY_DATABASE_LIST_T                tNewSlaves;
   RCX_VERIFY_DATABASE_LIST_T                tDeactivatedSlaves;
   RCX_VERIFY_SLAVE_DATABASE_LIST_T          tChangedSlaves;
   RCX_VERIFY_SLAVE_DATABASE_LIST_T          tUnchangedSlaves;
   RCX_VERIFY_SLAVE_DATABASE_LIST_T          tImpossibleSlaveChanges;
} RCX_VERIFY_DATABASE_CNF_DATA_T;

#define RCX_VERIFY_DATABASE_CNF_SIZE
   sizeof(RCX_VERIFY_DATABASE_CNF_DATA_T)


typedef RCX_VERIFY_DATABASE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                        tHead;    /* packet header */
  RCX_VERIFY_DATABASE_CNF_DATA_T             tData;    /* packet data */
} RCX_VERIFY_DATABASE_CNF_T;

#define RCX_VERFIY_DATABASE_CNF_PACKET_SIZE sizeof(RCX_VERIFY_DATABASE_CNF_T)
```

Packet Description

structure RCX_VERIFY_DATABASE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32		Destination end point identifier, unchanged
ulSrcId	UINT32		Source end point identifier, unchanged
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		
ulCmd	UINT32	0x2F83	RCX_VERIFY_DATABASE_CNF - Command
ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change
structure RCX_VERIFY_DATABASE_CNF_DATA_T			
tNewSlaves	STRUCT		Bit field which contains the addresses of the new slaves which have to be configured
tDeactivatedSlaves	STRUCT		Bit field which contains the addresses of the slaves which deactivated, or cannot be configured
tChangedSlaves	STRUCT		Bit field which contains the addresses of the slaves whose configuration has been changed
tUnchangedSlaves	STRUCT		Bit field which contains the addresses of the slaves whose configuration has not been changed
tImpossibleSlaveChanges	STRUCT		Bit field which contains the addresses of the slaves whose configuration is not valid

Table 198: RCX_VERIFY_DATABASE_CNF – Confirmation of Verify the new Configuration Database

6.5.2 RCX_ACTIVATE_DATABASE_REQ/CNF – Activate the new Configuration Database

This packet indicates the master to activate the new configuration.

This causes the following consequences at the slaves:

- Slaves mentioned in the old configuration file, which are not any longer mentioned in the new configuration file, will be removed from the bus.
- Slaves, which have been added to the configuration, will be parameterized and set to active state.
- Slaves, whose configuration has been changed will be deactivated, parameterized and activated.

Packet Structure Reference

```
typedef struct  RCX_ACTIVATE_DATABASE_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;              /* packet header */
} RCX_ACTIVATE_DATABASE_REQ_T;
```

Packet Description

structure RCX_ACTIVATE_DATABASE_REQ			Type: Request
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ `rcX queue handle`	Destination queue handle
ulSrc	UINT32	0 ... 2^{32}-1	Source queue handle
ulDestId	UINT32		Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	
ulCmd	UINT32	0x2F84	`RCX_ACTIVATE_DATABASE_REQ` - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change

Table 199: RCX_ACTIVATE_DATABASE_REQ – Activate the new Configuration Database

```
typedef struct RCX_ACTIVATE_DATABASE_CNF_DATA_Ttag
{
  TLR_UINT8 abSlvSt[16];              /* State of the slaves after configuration */
} RCX_ACTIVATE_DATABASE_CNF_DATA_T;

#define RCX_ACTIVATE_DATABASE_CNF_DATA_SIZE sizeof(RCX_ACTIVATE_DATABASE_CNF_DATA_T)


typedef struct  RCX_ACTIVATE_DATABASE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                  tHead;      /* packet header */
  RCX_ACTIVATE_DATABASE_CNF_DATA_T     tData;      /* packet data */
} RCX_ACTIVATE_DATABASE_CNF_T;
```

Packet Description

structure RCX_ACTIVATE_DATABASE_CNF			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle
ulSrc	UINT32		Source queue handle
ulDestId	UINT32		Destination end point identifier, specifying the final receiver of the packet within the destination process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source end point identifier, specifying the origin of the packet inside the source process
ulLen	UINT32	1	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	
ulCmd	UINT32	0x2F85	RCX_ACTIVATE_DATABASE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not change
structure RCX_ACTIVATE_DATABASE_CNF_DATA_T			
abSlvSt[16]	UINT8		Bit field which contains the addresses of slaves which are configured

Table 200: RCX_ACTIVATE_DATABASE_CNF - Confirmation of Activate the new Configuration Database

6.6 Handshake Mode

The PROFIBUS-DP Master stack offers several possibilities to exchange I/O data with the host application. In the most cases the host application runs independently from the bus cycle and a buffered mode is used. On the other hand, there are applications that need a synchronized I/O exchange with application. To achieve this requirement, the mode of I/O exchange can be configured.

The PROFIBUS DP Master stack supports the following modes:

Mode name	Input Handshake Mode	Output Handshake Mode	Sync Handshake Mode
buffered	host-controlled	host-controlled	not used
bus synchronous	host-controlled	host-controlled	not used

Table 201: Supported modes of PROFIBUS-DP Master protocol stack

Former versions of the stack also supported a device-controlled handshake mode for both data modes. Do not use device-controlled handshake mode for input anymore as the support for this mode will be discontinued!

6.6.1 Buffered / Input and Output Host-Controlled Mode

For the communication of the I/O data, a buffer is used that contains the I/O image for the host application and for the PROFIBUS DP Master stack. Both tasks can run independently having access to consistent I/O data. There is no relationship between the host application cycle and the PROFIBUS cycle. The PROFIBUS DP Master performs the data exchange with the slaves as fast as possible. If a PROFIBUS cycle is finished, a new PROFIBUS cycle will be started immediately if there is no violation of the minimum slave interval. Otherwise the minimum slave interval will be waited before starting new PROFIBUS cycle.

A host application exchanges I/O data over the dual-ported memory. To access the dual-ported memory, handshake flags are used. There is one handshake cell for the netX and one handshake cell for the host application. The bit `PDx_IN` is used for the input data and the bit `PDx_OUT` for the output data.

In output direction, the host application provides the new output data to the PROFIBUS DP Master by toggling the handshake flags. In input direction, the host application requests the latest input data image from the PROFIBUS DP Master. For more information on the handshake mechanism see reference [1].

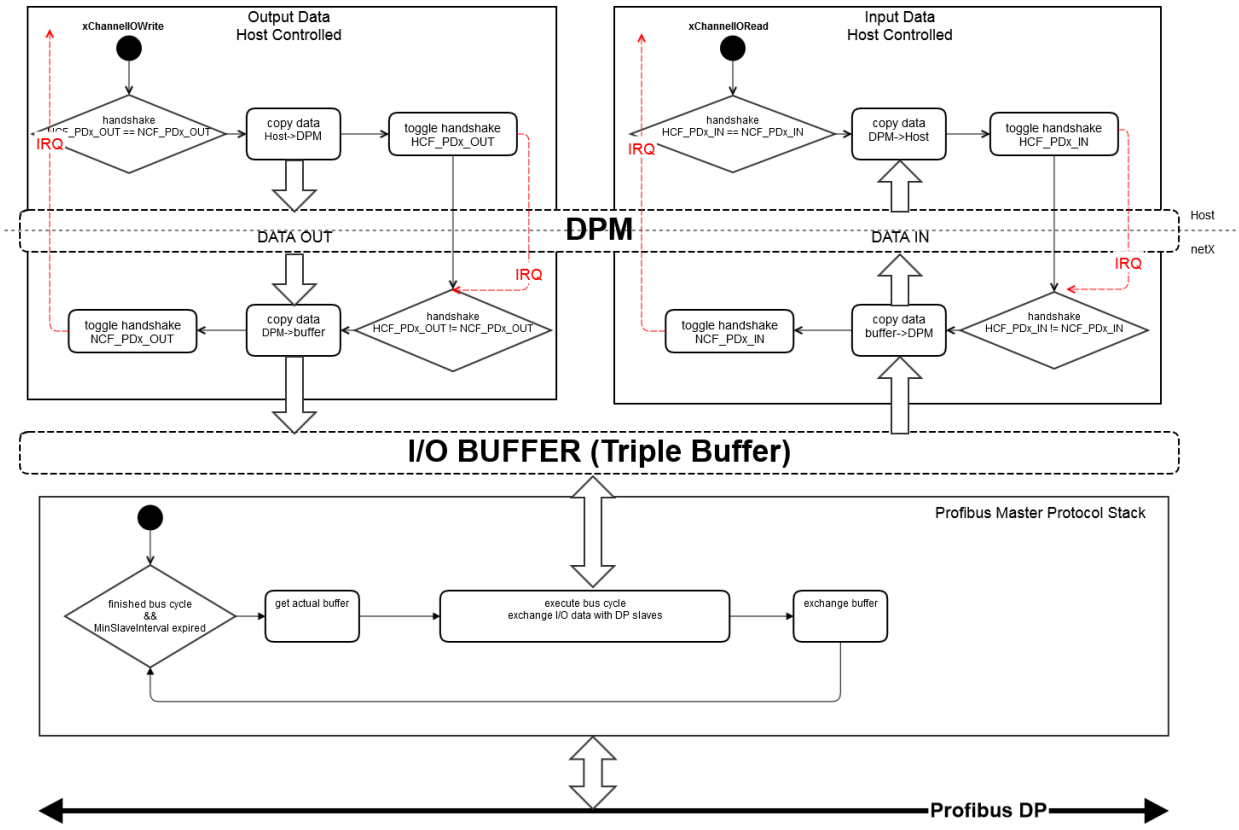

Figure 12: Buffered / Input and Output Host-Controlled Mode

6.6.2 Bus-synchronous / Input and Output Host-Controlled Mode

On bus-synchronous communication the host application controls the PROFIBUS cycle. When the host toggles the output handshake bit, output data will be copied into the internal memory and an output update event will be signalled to the PROFIBUS DP Master stack. The PROFIBUS DP Master checks for the right state and starts a PROFIBUS cycle. By finishing I/O exchange with all slaves the Master signals an Input Update Event. This event starts updating the input area of the dual-ported memory and changing the input handshake flag. The host application will be informed about the new input data and the end of the actual bus cycle by interrupt. The next PROFIBUS cycle will be started with the next handshake of output data from the host.

In this mode, the output handshake is used as sync source for triggering (the sync handshake is not used), see variable `bSyncSource` in section *Configuration* on page 274.

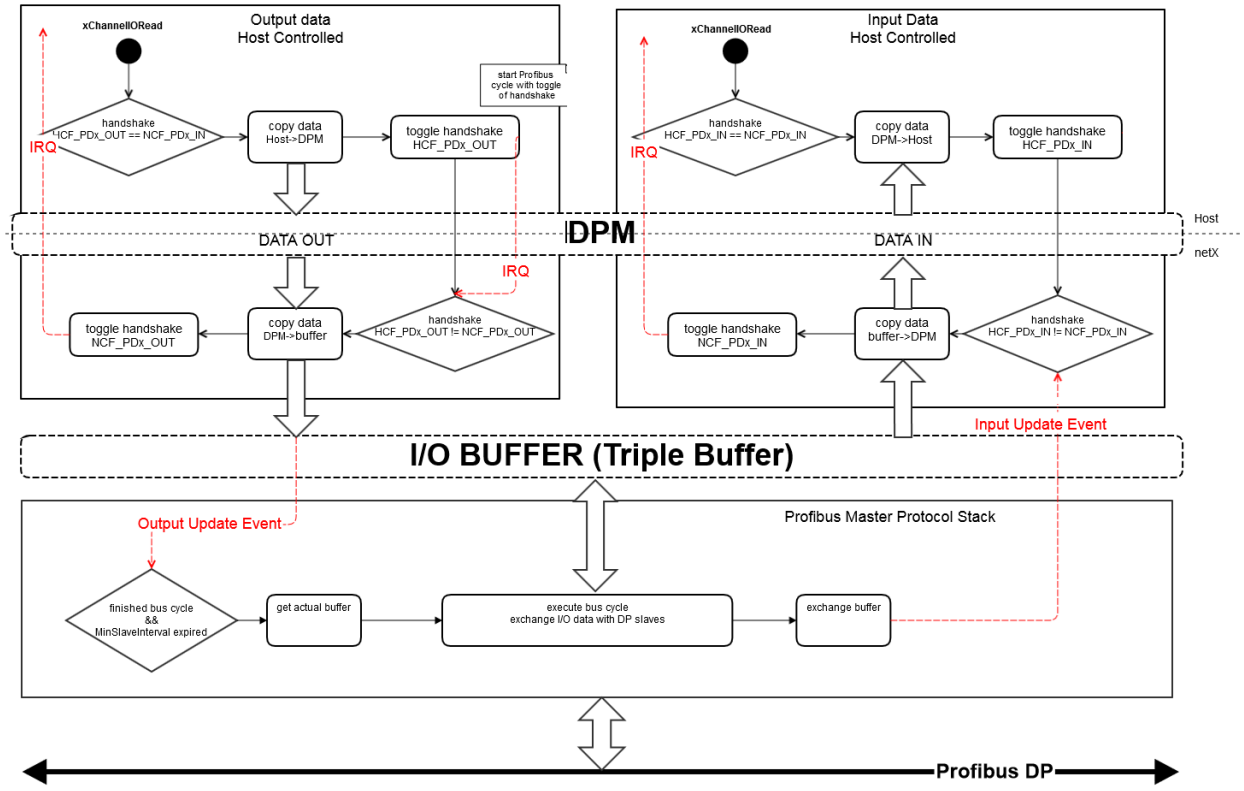

Figure 13: Bus Synchronous / Input and Output Host-Controlled Mode

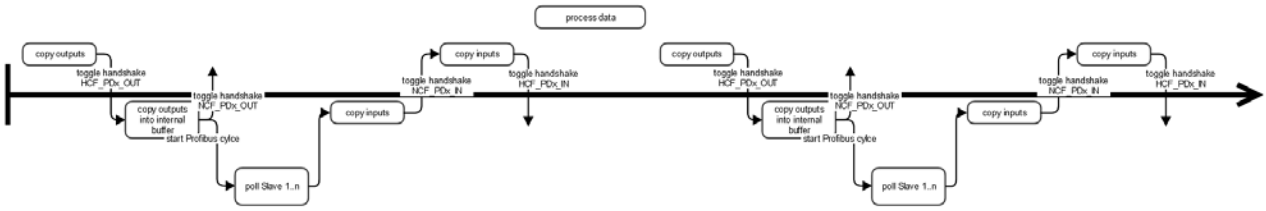

Figure 14: Handshake Timeline

6.6.3 Configuration

The handshake mode can be configured using the *Set Handshake Configuration Request* (0x2F34), documented in reference [1].

The handshake mode can be changed as long the PROFIBUS DP Master device is not in running state. In running state, this request will be rejected.

The following parameters are available for the PROFIBUS DP Master:

Variable	Values	Remark
bPDInHskMode	RCX_IO_MODE_DEFAULT (0) RCX_IO_MODE_BUFF_HST_CTRL (4)	Always host-controlled mode RCX_IO_MODE_DEFAULT sets value to RCX_IO_MODE_BUFF_HST_CTRL
bPDInSource	0	Always 0
usPDInErrorTh	0	Always 0
bPDOutHskMode	RCX_IO_MODE_DEFAULT (0) RCX_IO_MODE_BUFF_HST_CTRL (4)	Always host-controlled mode RCX_IO_MODE_DEFAULT sets value to RCX_SYNC_MODE_HST_CTRL
bPDOutSource	0	Always 0
usPDOutErrorTh	0	Always 0
bSyncHskMode	RCX_SYNC_MODE_OFF (0)	
bSyncSource	RCX_SYNC_SOURCE_OFF (0)	Mode as described in section *Buffered / Input and Output Host-Controlled Mode* on page 272.
	RCX_SYNC_SOURCE_1 (1)	Output handshake is used as sync source to trigger the bus cycle. Mode as described in section *Bus-synchronous / Input and Output Host-Controlled Mode* on page 273.
usSyncErrorTh	0	Always 0

Table 202: Handshake Parameters for PROFIBUS DP Master

Figure 15 shows the configuration states when the PROFIBUS Master is configured by pakets.

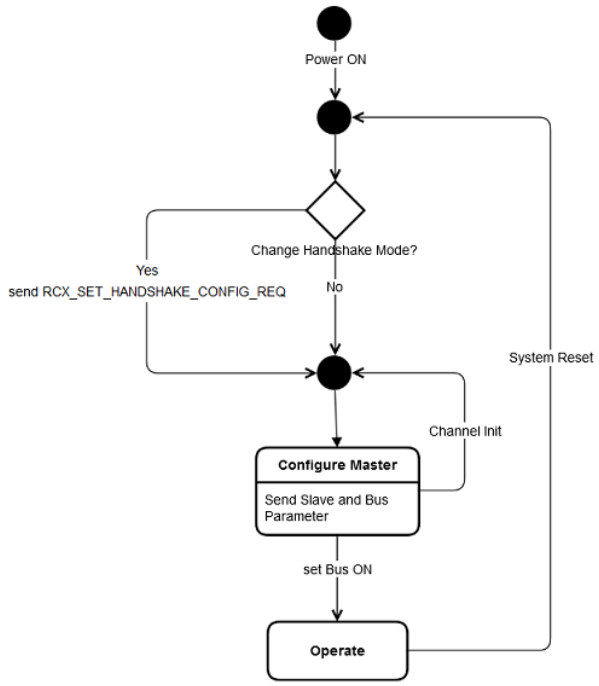

Figure 15: Configuration States (when using Packets to Configure the Master)

6.7 Network Scan

Multiple packets are necessary to perform a network scan. The API for network scan has a general part and PROFIBUS specific part. The general part is described in reference [5]. The PROFIBUS specific part is described below.

First, use RCX_BUSSCAN_REQ and RCX_BUSSCAN_CNF as described reference [5].

Use RCX_GET_DEVICE_INFO_REQ and RCX_GET_DEVICE_INFO_CNF as described reference [5]. The RCX_GET_DEVICE_INFO_CNF packet returns the PROFIBUS-specific ulStructId with value 3074. This value of ulStructId means that structure PROFIBUS_DL_DEVICE_INFO_T is used.

```
#define PROFIBUS_DL_DEVICE_INFO_STRUCT_ID 3074

typedef __TLR_PACKED_PRE struct PROFIBUS_DL_DEVICE_INFO_Ttag {
  UINT32 ulDeviceAdr;
} __TLR_PACKED_POST PROFIBUS_DL_DEVICE_INFO_T;
```

The device index from the bit list returned with the RCX_BUSSCAN_CNF corresponds with the PROFIBUS Station Address of a slave. This means that the RCX_GET_DEVICE_INFO_CNF packet returns again the Station address of the slave, which is no new information. The RCX_GET_DEVICE_INFO_CNF packet is provided to reach a generalized packet sequence for the network scan for several protocols. For PROFINET, this packet returns the Name of Station for example.

7 Topics for Linkable Object Module Users

7.1 Accessing the Protocol Stack by Programming the AP Task's Queue

In general, programming the AP task or the stack has to be performed according to the rules explained in the Hilscher Task Layer Reference Manual. There you can also find more information about the variables discussed in the following.

Getting the Receiver Task Handle of the Process Queue

To get the handle of the process queue of one of the various tasks the Macro `TLR_QUE_IDENTIFY()` needs to be used. It is described in detail within section 10.1.9.3 of the Hilscher Task Layer Reference Model Manual. This macro delivers a pointer to the handle of the intended queue to be accessed (which is returned within the third parameter, `phQue`), if you provide it with the name of the queue (and an instance of your own task). The correct ASCII-queue name for accessing a task which you have to use as current value for the first parameter (`pszIdn`) is

ASCII queue name	Description
`"FSPMM_QUE"`	Name of the FSPMM-Task process queue
`"FSPMM2_QUE"`	Name of the FSPMM2-Task process queue
`"PB_DL_QUE"`	Name of the DL-Task process queue
`"APM_QUE"`	Name of the APM-Task process queue

Table 203: Names of Queues in PROFIBUS DP Master Firmware

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the FSPMM_QUE-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the respective task.

8 Status/Error Codes Overview

8.1 Common Error Codes

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0000001	TLR_E_FAIL Common error, detailed error information optionally present in the data area of packet
0xC0000002	TLR_E_UNEXPECTED Unexpected failure.
0xC0000003	TLR_E_OUTOFMEMORY Ran out of memory.
0xC0000004	TLR_E_UNKNOWN_COMMAND Unknown Command in Packet received.
0xC0000005	TLR_E_UNKNOWN_DESTINATION Unknown Destination in Packet received.
0xC0000006	TLR_E_UNKNOWN_DESTINATION_ID Unknown Destination Id in Packet received.
0xC0000007	TLR_E_INVALID_PACKET_LEN Invalid packet length.
0xC0000008	TLR_E_INVALID_EXTENSION Invalid Extension in Packet received.
0xC0000009	TLR_E_INVALID_PARAMETER Invalid Parameter in Packet found.
0xC000000A	TLR_E_INVALID_ALIGNMENT Invalid alignment.
0xC000000C	TLR_E_WATCHDOG_TIMEOUT Watchdog error occurred.
0xC000000D	TLR_E_INVALID_LIST_TYPE List type is invalid.
0xC000000E	TLR_E_UNKNOWN_HANDLE Handle is unknown.
0xC000000F	TLR_E_PACKET_OUT_OF_SEQ A packet index has been not in the expected sequence.
0xC0000010	TLR_E_PACKET_OUT_OF_MEMORY The amount of fragmented data contained the packet sequence has been too large.
0xC0000011	TLR_E_QUE_PACKETDONE The packet done function has failed.
0xC0000012	TLR_E_QUE_SENDPACKET The sending of a packet has failed.
0xC0000013	TLR_E_POOL_PACKET_GET The request of a packet from packet pool has failed.
0xC0000014	TLR_E_POOL_PACKET_RELEASE The release of a packet to packet pool has failed.
0xC0000015	TLR_E_POOL_GET_LOAD The get packet pool load function has failed.

Hexadecimal Value	Definition Description
0xC0000016	TLR_E_QUE_GET_LOAD The get queue load function has failed.
0xC0000017	TLR_E_QUE_WAITFORPACKET The waiting for a packet from queue has failed.
0xC0000018	TLR_E_QUE_POSTPACKET The posting of a packet has failed.
0xC0000019	TLR_E_QUE_PEEKPACKET The peek of a packet from queue has failed.
0xC000001A	TLR_E_REQUEST_RUNNING A new request was received, but one request is still running.
0xC000001B	TLR_E_CREATE_TIMER Creating a timer failed.
0xC0000100	TLR_E_INIT_FAULT General initialization fault.
0xC0000101	TLR_E_DATABASE_ACCESS_FAILED Database couldn't be opened. No verification possible.
0xC0000102	TLR_E_CIR_MASTER_PARAMETER_FAILED Parameters of the master were changed, which is not allowed.
0xC0000103	TLR_E_CIR_SLAVE_PARAMETER_FAILED Result of the verification: One or more slave parameter set have an error.
0xC0000119	TLR_E_NOT_CONFIGURED Configuration not available.
0xC0000120	TLR_E_CONFIGURATION_FAULT General configuration fault.
0xC0000121	TLR_E_INCONSISTENT_DATA_SET Inconsistent configuration data.
0xC0000122	TLR_E_DATA_SET_MISMATCH Configuration data set mismatch.
0xC0000123	TLR_E_INSUFFICIENT_LICENSE Insufficient license.
0xC0000124	TLR_E_PARAMETER_ERROR Parameter error.
0xC0000125	TLR_E_INVALID_NETWORK_ADDRESS Network address invalid.
0xC0000126	TLR_E_NO_SECURITY_MEMORY Security memory chip missing or broken.
0xC0000140	TLR_E_NETWORK_FAULT General communication fault.
0xC0000141	TLR_E_CONNECTION_CLOSED Connection closed.
0xC0000142	TLR_E_CONNECTION_TIMEOUT Connection timeout.
0xC0000143	TLR_E_LONELY_NETWORK Lonely network.
0xC0000144	TLR_E_DUPLICATE_NODE Duplicate network address.

Hexadecimal Value	Definition Description
0xC0000145	TLR_E_CABLE_DISCONNECT Cable disconnected.
0xC0000180	TLR_E_BUS_OFF Bus Off flag is set.
0xC0000181	TLR_E_CONFIG_LOCK Changing configuration is not allowed.
0xC0000182	TLR_E_APPLICATION_NOT_READY Application is not at ready state.
0xC0000183	TLR_E_RESET_IN_PROCESS Application is performing a reset.
0xC0000200	TLR_E_WATCHDOG_TIME_INVALID Watchdog time is out of range.
0xC0000201	TLR_E_APPLICATION_ALREADY_REGISTERED Application is already registered.
0xC0000202	TLR_E_NO_APPLICATION_REGISTERED No application registered.

Table 204: Common Error codes

8.2 Error Codes of the FSPMM-Task

Hexadecimal Value	Definition Description
0xC0380001	TLR_E_PROFIBUS_FSPMM_COMMAND_INVALID Invalid command received.
0xC0380002	TLR_E_PROFIBUS_FSPMM_INV_BUSMODE Invalid bus mode for this command.
0xC0380003	TLR_E_PROFIBUS_FSPMM_RESET FSPMM task is reset.
0xC0380004	TLR_E_PROFIBUS_FSPMM_ACLR PROFIBUS Master is at auto clear state.
0xC0380005	TLR_E_PROFIBUS_FSPMM_CONTROL_TIMER_EXPIRED Data Control Timer expired. No bus access for sending global control.
0xC0380006	TLR_E_PROFIBUS_FSPMM_ALARM_OVERFLOW Alarm buffer overflow.
0xC0380007	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_INIT Alarm handler is not initialized.
0xC0380008	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_STARTED Alarm handler is not started.
0xC0380009	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_ENABLED Alarms are disabled.
0xC038000A	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_PENDING Alarm is not at a pending state.
0xC038000B	TLR_E_PROFIBUS_FSPMM_ALARM_STATE_ERROR Invalid Alarm state.
0xC038000C	TLR_E_PROFIBUS_FSPMM_ALARM_SEQ_ERROR Alarm sequence error occurred.
0xC038000D	TLR_E_PROFIBUS_FSPMM_MSAC1_STATE_ERROR Alarm handler is not at the proper state.
0xC038000E	TLR_E_PROFIBUS_FSPMM_MSAC1_FAULT Get an alarm acknowledge without an alarm.
0xC038000F	TLR_E_PROFIBUS_FSPMM_INVALID_AREA_CODE Invalid area code or slave address received.
0xC0380011	TLR_E_PROFIBUS_FSPMM_IV_DL_DATA_LEN Invalid data length.
0xC0380012	TLR_E_PROFIBUS_FSPMM_IV_BUS_PRM Invalid bus parameter received.
0xC0380013	TLR_E_PROFIBUS_FSPMM_IV_SLAVE_PRM Invalid slave parameter received.
0xC0380014	TLR_E_PROFIBUS_FSPMM_ACK_NO Command cannot be executed at the actual bus state.
0xC0380015	TLR_E_PROFIBUS_FSPMM_ACK_GE Error while sending global control.
0xC0380016	TLR_E_PROFIBUS_FSPMM_MSAL1_FAULT Failure at alarm handler. Alarm handler is stopped.
0xC0380017	TLR_E_PROFIBUS_FSPMM_MSAC2_FAULT Failure at MSAC2 handler.

Hexadecimal Value	Definition Description
0xC0380018	TLR_E_PROFIBUS_FSPMM_REJ_SE Device is stopping the communication or not in OPEN state.
0xC0380019	TLR_E_PROFIBUS_FSPMM_REJ_PS A previous service is still in process.
0xC038001A	TLR_E_PROFIBUS_FSPMM_REJ_LE Message has an invalid length.
0xC038001B	TLR_E_PROFIBUS_FSPMM_REJ_IV Invalid parameter at request.
0xC038001C	TLR_E_PROFIBUS_FSPMM_REJ_ABORT Device aborts DP V1 communication.
0xC038001D	TLR_E_PROFIBUS_FSPMM_INVALID_SLAVE_ADDRESS Invalid slave address.
0xC038001E	TLR_E_PROFIBUS_FSPMM_ALREADY_INITIALZED FSPMM Already initialized.
0xC038001F	TLR_E_PROFIBUS_FSPMM_INVALID_APPLICATION Command from not registered application.
0xC0380020	TLR_E_PROFIBUS_FSPMM_DMPMM_IV_STATE Command not allowed in current state.
0xC0380021	TLR_E_PROFIBUS_FSPMM_PB_FLAG_ERROR_ACTION_FLAG Function 'AUTO CLEAR' not supported.
0xC0380022	TLR_E_PROFIBUS_FSPMM_PB_FLAG_ISO_MODE_MSK Function 'ISO_MODE' not supported.
0xC0380023	TLR_E_PROFIBUS_FSPMM_DL_PB_FLAG_ISOM_SYNC Function 'ISO_MODE_SYNC' not supported.
0xC0380024	TLR_E_PROFIBUS_FSPMM_DL_PB_FLAG_ISOM_FREEZE Function 'ISO_MODE_FREEZE' not supported.
0xC0380025	TLR_E_PROFIBUS_FSPMM_MSAC1_NRS Negative response received.
0xC0380026	TLR_E_PROFIBUS_FSPMM_MSALM_ALARM_ACK_NEG Negative alarm acknowledge response received.

Table 205: Error Codes of the FSPMM-Task

8.2.1 Diagnostic Codes of the FSPMM-Task

Hexadecimal Value	Definition Description
0xC0380001	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_NO_INIT_PACKET Getting a packet from the pool to initialize the Data Link failed.
0xC0380002	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_BUS_PARAMETER_REJECTED Command for setting the bus parameter is rejected.
0xC0380003	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_SSAP_ACTIVATE_FAIL Activating the source SAP failed.
0xC0380004	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_STATE_ERROR DMPMM state machine is at an invalid state.
0xC0380005	TLR_DIAG_E_PROFIBUS_FSPMM_FSPMM_STATE_ERROR FSPMM state machine is at an invalid state.
0xC0380006	TLR_DIAG_E_PROFIBUS_FSPMM_MSCY1M_STATE_ERROR MSCY1M state machine is at an invalid state.
0xC0380007	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_SSAP_ACTIVATE_RESPONDER_FAIL Activate RSAP failed.
0xC0380008	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_NO_GLBCNTRL_PACKET Getting a packet from the pool to send global control failed.
0xC0380009	TLR_DIAG_E_PROFIBUS_FSPMM_NOT_SUPPORTED Function is not supported at the current version of the stack.
0xC038000A	TLR_DIAG_E_PROFIBUS_FSPMM_NO_DATABASE No database found.
0xC038000B	TLR_DIAG_E_PROFIBUS_FSPMM_INVALID_DATABASE Getting a packet from the pool to initialize the Master state machine failed.
0xC038000C	TLR_DIAG_E_PROFIBUS_FSPMM_UNKNOWN_TELEGRAM Telegram with an unknown SAP was received.
0xC038000E	TLR_DIAG_E_PROFIBUS_FSPMM_UNKNOWN_DPV1_CMD Answer of an invalid DP V1 service received.
0xC0380010	TLR_DIAG_E_PROFIBUS_FSPMM_TASK_PRM_VERSION_INVALID Invalid version of Task Parameters.

Table 206: Diagnostic Codes of the FSPMM-Task

8.3 Error Codes of the FSPMM2-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0690000	TLR_E_PROFIBUS_FSPMM2_COMMAND_INVALID Invalid command received.
0xC0690001	TLR_E_PROFIBUS_FSPMM2_LENGTH_INVALID Invalid data length.
0xC0690002	TLR_E_PROFIBUS_FSPMM2_NOT_IMPLEMENTED Service not implemented.
0xC0690003	TLR_E_PROFIBUS_FSPMM2_ADD_INVALID Invalid address.
0xC0690004	TLR_E_PROFIBUS_FSPMM2_SERVICE_IN_REQUEST Previous service already in request.
0xC0690005	TLR_E_PROFIBUS_FSPMM2_NOT_IN_OPEN_STATE Connection is not in state open.
0xC0690006	TLR_E_PROFIBUS_FSPMM2_OUT_OF_RESOURCES Out of resources for new connections.
0xC0690007	TLR_E_PROFIBUS_FSPMM2_IN_USE Connection to this slave already in use.
0xC0690008	TLR_E_PROFIBUS_FSPMM2_ALREADY_INIT Stack is already initialized.
0xC0690009	TLR_E_PROFIBUS_FSPMM2_COM_REFERENCE_INVALID Invalid communication reference.
0xC069000A	TLR_E_PROFIBUS_FSPMM2_TIMEOUT Timeout error.
0xC069000B	TLR_E_PROFIBUS_FSPMM2_INITIATE_ABT_STO Parameter Send Timeout to small.
0xC069000C	TLR_E_PROFIBUS_FSPMM2_INITIATE_ABT_FE Telegram format error.
0xC069000D	TLR_E_PROFIBUS_FSPMM2_NRS Negative response.
0xC069000E	TLR_E_PROFIBUS_FSPMM2_ABORT Service Aborted.
0xC0690100	TLR_E_PROFIBUS_FSPMM2_CON_XX Service confirmation negative.

Table 207: Error Codes of the FSPMM-Task

8.3.1 Diagnostic Codes of the FSPMM2-Task

Currently no diagnostic codes are defined for the FSPMM2-task.

8.4 Error Codes of the DL-Task

Hexadecimal Value	Definition Description
0xC0060001	TLR_E_PROFIBUS_DL_COMMAND_INVALID Invalid command received.
0xC0060040	TLR_E_PROFIBUS_DL_XC_INVALID The assigned XC-Data Link Layer is not installed or has a pending error.
0xC0060041	TLR_E_PROFIBUS_DL_BAUDRATE_INVALID The specified baud rate option is not supported and is out of range.
0xC0060042	TLR_E_PROFIBUS_DL_GAP_UPDATE_INVALID The specified GAP update factor option is not supported and is out of range 1-100.
0xC0060043	TLR_E_PROFIBUS_DL_DL_ADDR_INVALID The specified local PROFIBUS address option is not supported and is out of range 0-125.
0xC0060044	TLR_E_PROFIBUS_DL_RETRY_LIMIT The specified retry limit option is not supported and is zero.
0xC0060045	TLR_E_PROFIBUS_DL_HSA_INVALID The specified highest station address option is not supported and is out of range 0-126.
0xC0060046	TLR_E_PROFIBUS_DL_NO_BUS_PARAMETER_SET The service cannot be executed, there are no bus parameters specified yet.
0xC0060047	TLR_E_PROFIBUS_DL_DLE_NOT_RESPONDING The service has detected a timeout at the connected XC-Data Link Layer entity.
0xC0060048	TLR_E_PROFIBUS_DL_NO_DL_RESOURCE There are no further resource blocks available to execute the service within the connected XC-Data Link Layer entity.
0xC0060049	TLR_E_PROFIBUS_DL_FATAL_DL_RESOURCE There are no further resource blocks available to execute the service within the connected XC-Data Link Layer entity.
0xC0060050	TLR_E_PROFIBUS_DL_STOPPED PROFIBUS is stopped command cannot be handled.
0xC0060051	TLR_E_PROFIBUS_DL_PENDING_PACKET Previous pending packet is returned. It could not be handled.
0xC0060052	TLR_E_PROFIBUS_DL_SLAVE_MODE Command could not be executed, DL-task is running at slave mode.
0xC0060080	TLR_E_PROFIBUS_DL_ACK_UE The remote station the service has been sent to indicates a User Error as service acknowledgement.
0xC0060081	TLR_E_PROFIBUS_DL_ACK_RR The remote station the service has been sent to indicates a Resource Error as service acknowledgement.
0xC0060082	TLR_E_PROFIBUS_DL_ACK_RS The remote station the service has been sent to indicates a Service Access Point Error as service acknowledgement.
0xC0060083	TLR_E_PROFIBUS_DL_ACK_NR The remote station the service has been sent to confirms its positive reception but has no data to confirm.
0xC0060084	TLR_E_PROFIBUS_DL_ACK_RDH The remote station the service has been sent to, confirms its reception negatively but has returned low priority data in the response.

Hexadecimal Value	Definition Description
0xC0060085	TLR_E_PROFIBUS_DL_ACK_RDL The remote station the service has been sent to, confirms its reception negatively but has returned high priority data in the response.
0xC0060086	TLR_E_PROFIBUS_DL_ACK_DH The remote station the service has been sent to, confirms its reception positively and has returned low priority data in the response.
0xC0060087	TLR_E_PROFIBUS_DL_ACK_DL The remote station the service has been sent to, confirms its reception positively and has returned high priority data in the response.
0xC0060088	TLR_E_PROFIBUS_DL_ACK_NA The remote station the service has been sent to shows no or no plausible reaction at all.
0xC0060089	TLR_E_PROFIBUS_DL_ACK_UNKNOWN The remote station the service has been sent has returned an unknown acknowledgement code.
0xC006008A	TLR_E_PROFIBUS_DL_ACK_LS The requested service is not activated within the local SAP configuration.
0xC006008B	TLR_E_PROFIBUS_DL_ACK_LR The local resources needed to execute the requested service are not available or not sufficient.
0xC006008C	TLR_E_PROFIBUS_DL_ACK_DS The local data link layer is not in the logical token ring or disconnected from the network.
0xC006008D	TLR_E_PROFIBUS_DL_ACK_IV Invalid parameter detected in the requested service.
0xC006008E	TLR_E_PROFIBUS_DL_ACK_NO The local SAP is not activated because it has been activated already or resources are not sufficient.
0xC006008F	TLR_E_PROFIBUS_DL_ACK_NO_SET The variable to be set does not exist.
0xC0060090	TLR_E_PROFIBUS_DL_ACK_RE Format error of the telegram.
0xC0060091	TLR_E_PROFIBUS_DL_TSET_INVALID The specified parameter TSET is out of range 1-255.

Table 208: Error Codes of the DL-Task

8.4.1 Diagnostic Codes of the DL-Task

Hexadecimal Value	Definition Description
0xC0060001	TLR_DIAG_E_PROFIBUS_DL_DATA_ACK_RES Invalid "Data Ack (SDA)" response packet received. Response does not match to internally reserved communication block
0xC0060002	TLR_DIAG_E_PROFIBUS_DL_DATA_RES Invalid "Data (SDN)" response packet received. Response does not match to internally reserved communication block
0xC0060003	TLR_DIAG_E_PROFIBUS_DL_DATA_REPLY_RES Invalid "Data Reply (SRD)" response packet received. Response does not match to internally reserved communication block

Table 209: Diagnostic Codes of the DL-Task

8.5 Error Codes of the APM-Task

Hexadecimal Value	Definition Description
0xC0390001	TLR_E_PROFIBUS_APM_COMMAND_INVALID Invalid command received.
0xC0390002	TLR_E_PROFIBUS_APM_COMMAND_ALREADY_IN_RQUEST Command already in request.
0xC0390003	TLR_E_PROFIBUS_APM_TIO_RESET_W_MODE_STOP Timeout while stopping PROFIBUS.
0xC0390004	TLR_E_PROFIBUS_APM_TIO_RESET_W_INIT_FSPMM Timeout while resetting PROFIBUS.
0xC0390005	TLR_E_PROFIBUS_APM_NON_EXCHANGE_SLAVE No data exchange with at least one slave.
0xC0390006	TLR_E_PROFIBUS_APM_NON_EXCHANGE_ALL No slave in data exchange.
0xC0390007	TLR_E_PROFIBUS_APM_CONFIG_LOCK Configuration locked.
0xC0390008	TLR_E_PROFIBUS_APM_CONFIG_VIA_DBM Already configured via data base.
0xC0390009	TLR_E_PROFIBUS_APM_ALREADY_CONFIGURED Already configured.
0xC039000A	TLR_E_PROFIBUS_APM_CHANNEL_INIT_IN_PROGRESS Channel initialization in progress.
0xC039000B	TLR_E_PROFIBUS_APM_CHANNEL_INIT_FAILED Channel initialization failed.

Table 210: Error Codes of the APM-Task

8.5.1 Diagnostic Codes of the APM-Task

Hexadecimal Value	Definition Description
0xC0390032	TLR_DIAG_E_PROFIBUS_APM_NO_FSPMM No FSPMM-Task or -queue found to communicate.
0xC039003A	TLR_DIAG_E_PROFIBUS_ADR_DOUBLE Double station address detected.
0xC039003B	TLR_DIAG_E_PROFIBUS_MAX_OUTPUT_OFFSET Maximum output offset reached.
0xC039003C	TLR_DIAG_E_PROFIBUS_MAX_INPUT_OFFSET Maximum input offset reached
0xC039003D	TLR_DIAG_E_PROFIBUS_OUTPUT_OVERLAP Overlap in output data.
0xC039003E	TLR_DIAG_E_PROFIBUS_INPUT_OVERLAP Overlap in input data.
0xC03900DC	TLR_DIAG_E_PROFIBUS_APM_WATCHDOG_ERROR Watchdog Error.
0xC03900DE	TLR_DIAG_E_PROFIBUS_APM_AUTOCLEAR_ERROR Auto-clear activated.
0xC03900E1	TLR_DIAG_E_PROFIBUS_APM_OUT_OF_PACKET No packet left to communicate.
0xC0390100	TLR_DIAG_E_PROFIBUS_APM_NO_INIT_PACKET Getting a packet from the pool to initialize the Master state machine failed.
0xC0390101	TLR_DIAG_E_PROFIBUS_APM_NO_SET_MODE_PACKET Getting a packet from the pool to set the Master mode failed.
0xC0390102	TLR_DIAG_E_PROFIBUS_APM_NO_SET_OUTPUT_PACKET Getting a packet from the pool to set output date fail.
0xC0390103	TLR_DIAG_E_PROFIBUS_APM_NO_GET_INPUT_PACKET Getting a packet from the pool to request input data failed.
0xC0390104	TLR_DIAG_E_PROFIBUS_APM_GET_INPUT_FAIL Getting input data fail.
0xC0390105	TLR_DIAG_E_PROFIBUS_APM_SET_OUTPUT_FAIL Write Slave output data failed.

Table 211: Diagnostic Codes of the APM-Task

9 Appendix

9.1 Legal Notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

9.2 List of Figures

Figure 1: Internal Structure of PROFIBUS DP-Master Firmware 10
Figure 2: PROFIBUS in the OSI/ISO Layer Model 13
Figure 3: Initialization Sequence of DP V1 Class2-Connection 35
Figure 4: Profibus Master DP V1 Class2 Initialization 36
Figure 5: Profibus Master DP V1 Class2 Services 40
Figure 6: Profibus Master DP V1 Class2 Close/Abort 41
Figure 7: SDA Service with Acknowledgement 44
Figure 8: Service Access Points 46
Figure 9: Configuration during Network State OPERATE 51
Figure 10: Redundancy Functionality – Host Application 53
Figure 11: Arrangement of Input and Output Data 69
Figure 12: Buffered / Input and Output Host-Controlled Mode 272
Figure 13: Bus Synchronous / Input and Output Host-Controlled Mode 273
Figure 14: Handshake Timeline 273
Figure 15: Configuration States (when using Packets to Configure the Master) 274

9.3 List of Tables

Table 1: List of Revisions 5
Table 2: Terms, Abbreviations and Definitions 9
Table 3: References to Documents 9
Table 4: PROFIBUS DP Operation Modes (States) 14
Table 5: Packets for DP V1 Class1 Acyclic Data Transfer 17
Table 6: Explanation of Error Class and Error Code within Error_Code_1 18
Table 7: General Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification 21
Table 8: Special Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification 22
Table 9: Structure of Length Byte in the Special Identifier Format of the Identifier Byte according to IEC 61158/ EN 50170 Specification 23
Table 10: Octet 1: Station Status_1 26
Table 11: Octet 2: Station Status_2 27
Table 12: Octet 3: Station Status_3 27
Table 13: Diagnosis Header 28
Table 14: Identification-related diagnosis – First Data Byte 28
Table 15: Identification-related diagnosis – Second Data Byte 29
Table 16: Channel-related Diagnosis – First Data Byte 30
Table 17: Channel-related Diagnosis – Second Data Byte 30
Table 18: Overview of available Packets supporting DP V1 Class2 31
Table 19:DP V1 Class2 Initialization Parameter 37
Table 20:DP V1 Class2 Initialization additional address parameter 37
Table 21:DP V1 Class2 Initialization format of address parameter 38
Table 22: Bus and Master Parameters 55
Table 23: Supported Baud Rates 57
Table 24:Slave Parameters, their Meanings and their Ranges of allowed Values 62
Table 25: Explanation of Bits of Sl_Flag within Slave Parameter Block 63
Table 26: Meaning of Slave_Type 64
Table 27: Meaning of Alarm_Mode Parameter 64
Table 28: Explanation of Bits of Sl_Flag within Slave Parameter Block 65
Table 29: Octet 1 of Prm_Data Slave Parameter 66
Table 30: Meaning of Combinations of Lock_Req and Unlock_Req Bits 66
Table 31: abAddTab Structure 68
Table 32: Extended Status Block 70
Table 33: Extended Status for PROFIBUS DP-Master 72
Table 34: Global Bits Variable 72
Table 35: Operation Modes of the PROFIBUS DP Master and their Values 73
Table 36: Relationship between Slave Station Address and the corresponding abSl_cfg Bit 74
Table 37: Relationship between Slave Station Address and the corresponding abSl_state Bit 74
Table 38: Relationship between Slave Station Address and the corresponding abSl_diag Bit 75
Table 39: Relationship between abSl_state and abSl_diag bits 75
Table 40: Extended Status Field 77
Table 41: FSPMM-Task process queue 79
Table 42: Overview over the Packets of the FSPMM-Master -Task of the PROFIBUS DP-Master Protocol Stack 80
Table 43: PROFIBUS_FSPMM_APP_REG_REQ – Application Register Request 81
Table 44: PROFIBUS_FSPMM_APP_REG_REQ– Packet Status/Error 81
Table 45: PROFIBUS_FSPMM_APP_REG_CNF – Application Register Confirmation 82

Table 46: PROFIBUS_FSPMM_APP_REG_CNF - Packet Status/Error ... 82
Table 47: PROFIBUS_FSPMM_CMD_INIT_REQ – Request for setting of Bus Parameters ... 84
Table 48: PROFIBUS_FSPMM_CMD_INIT_CNF - Confirmation Command for Request for Setting of Bus Parameter ... 85
Table 49: PROFIBUS_FSPMM_CMD_SET_MODE_REQ– Set a new operation mode ... 87
Table 50: PROFIBUS_FSPMM_CMD_SET_MODE_CNF – Confirmation to Set a new operation mode ... 88
Table 51: PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication ... 90
Table 52: ulFlags Bit Mask ... 91
Table 53: PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ – Request a Slave Diagnostic ... 92
Table 54: PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_CNF – Confirmation of Request a Slave Diagnostic ... 94
Table 55: PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic ... 95
Table 56: PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ– Set new Output Data ... 96
Table 57: PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF – Confirmation of Setting new Output Data ... 97
Table 58: PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF – Packet Status/Error ... 97
Table 59: PROFIBUS_FSPMM_CMD_GET_INPUT_REQ - Get new Input Data ... 98
Table 60: PROFIBUS_FSPMM_CMD_GET_INPUT_CNF - Confirmation of Get new Input Data ... 99
Table 61: PROFIBUS_FSPMM_CMD_READ_REQ –V1 Class1 Read Request ... 102
Table 62: PROFIBUS_FSPMM_CMD_READ_CNF –Confirmation of V1 Class1 Read Request ... 104
Table 63: PROFIBUS_FSPMM_CMD_WRITE_REQ– V1 Class1 Write Request ... 106
Table 64: PROFIBUS_FSPMM_CMD_WRITE_CNF – Confirmation of V1 Class1 Write Request ... 108
Table 65: PROFIBUS_FSPMM_CMD_WRITE_CNF – Packet Status/Error ... 109
Table 66: Alarm Types ... 110
Table 67: Flags of `Alarm_Spec_Ack` parameter ... 111
Table 68: Alarm_Specifier Bits D1 and D0 ... 111
Table 69: Add_Ack Bit D2 ... 111
Table 70: PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification ... 112
Table 71: Possible Errors and Descriptions ... 113
Table 72: PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ – Alarm Acknowledge ... 114
Table 73: PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF – Confirmation of Alarm Acknowledge ... 115
Table 74: PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF – Negative Confirmation of Alarm Acknowledge ... 117
Table 75: PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ– Download Slave Parameter Set ... 119
Table 76: Structure of the Slave Parameter Set according to IEC 61158 Specification ... 120
Table 77: `abAddTab` Structure ... 121
Table 78: Example of a Slave Parameter Data Set ... 123
Table 79: PROFIBUS_FSPMM_CMD_DOWNLOAD_CNF – Configuration of Download Slave Parameter Set ... 124
Table 80: Explanation of Bits in Global Control_Command ... 125
Table 81: Explanations of possible Combinations of the Bits 'Unsync'/'Sync' and 'Unfreeze'/'Freeze': ... 126
Table 82: GroupSelect ... 126
Table 83: PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ– Send a Global Control Request ... 127
Table 84: PROFIBUS_FSPMM_CMD_GLOBALCONTROL_CNF – Confirmation of Sending a Global Control Message ... 128
Table 85: PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ – Slave Activate Request Packet ... 129
Table 86: PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_CNF – Slave Activate Confirmation Packet ... 130
Table 87: PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a fatal Fault ... 131
Table 88: PROFIBUS_MSCS1M_CMD_INIT_CS_REQ – Initialize ClockSync Feature ... 132
Table 89: PROFIBUS_MSCS1M_CMD_INIT_CS_CNF – Confirmation to Initialize ClockSync Feature Request ... 133
Table 90: PROFIBUS_MSCS1M_CMD_SET_TIME_REQ – Set Time Request ... 135
Table 91: PROFIBUS_MSCS1M_CMD_SET_TIME_CNF – Confirmation to Set Time Request ... 136
Table 92: PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_REQ – Switch Redundancy ... 138
Table 93: PROFIBUS_MSCY1M_CMD_REDUNDANCY_SWITCH_CNF – Confirmation to Switch Redundancy Packet ... 139
Table 94: PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_REQ - Input Change of Slaves Request ... 140
Table 95: PROFIBUS_FSPMM_CMD_UPDATE_ICOS_CONFIG_CNF – Confirmation to Input Change of Slaves Request ... 141
Table 96: PROFIBUS_FSPMM_CMD_UPLOAD_REQ – Upload Slave Parameter Set Request ... 143
Table 97: PROFIBUS_FSPMM_CMD_UPLOAD_CNF – Confirmation of Upload Slave Parameter Set Request ... 144
Table 98: FSPMM-Task process queue ... 145
Table 99: Overview over the Packets of the FSPMM-Master -Task of the PROFIBUS DP-Master Protocol Stack ... 146
Table 100: PROFIBUS_FSPMM2_CMD_INIT_REQ- Initialization Command for Class2 Connection ... 147
Table 101: PROFIBUS_FSPMM2_CMD_INIT_CNF - Confirmation of Initialization Command ... 148
Table 102: PROFIBUS_FSPMM2_CMD_INITIATE_REQ - Initiate Command ... 153
Table 103: PROFIBUS_FSPMM2_CMD_INITIATE_CNF_POS - Positive Confirmation of Initiate Command ... 155
Table 104: PROFIBUS_FSPMM2_CMD_INITIATE_CNF_NEG - Negative Confirmation of Initiate Command ... 156
Table 105: PROFIBUS_FSPMM2_CMD_READ_REQ – Read Command ... 158
Table 106: PROFIBUS_FSPMM2_CMD_READ_CNF_POS - Positive Confirmation of Read Command ... 160
Table 107: PROFIBUS_FSPMM2_CMD_READ_CNF_POS - Packet Status/ErrorPacket Description ... 160
Table 108: PROFIBUS_FSPMM2_CMD_READ_CNF_NEG - Negative Confirmation of Read Command ... 161
Table 109: PROFIBUS_FSPMM2_CMD_WRITE_REQ – Write Command ... 163
Table 110: PROFIBUS_FSPMM2_CMD_WRITE_CNF_POS - Positive Confirmation of Write Command ... 165
Table 111: PROFIBUS_FSPMM2_CMD_WRITE_CNF_NEG - Negative Confirmation of Write Command ... 166

Table 112: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ – Data Transport Command 168
Table 113: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF_POS - Positive Confirmation of Data Transport Command 170
Table 114: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF_NEG - Negative Confirmation of Data Transport Command 171
Table 115: Allowed Values of Subnet Variable 172
Table 116: Instance Codes and their Meaning 172
Table 117: Possible Reason Codes and their Meaning 173
Table 118: PROFIBUS_FSPMM2_CMD_ABORT_REQ – Abort Request 174
Table 119: PROFIBUS_FSPMM2_CMD_ABORT_CNF - Confirmation of Abort Request 175
Table 120: PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ - Read Slave Diagnostics (DP V1 Class2) 176
Table 121: PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_CNF - Confirmation of Read Slave Diagnostics (DP V1 Class2) 177
Table 122: PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ – Read Input Request 178
Table 123: PROFIBUS_FSPMM2_CMD_READ_INPUT_CNF – Confirmation of Read Input 179
Table 124: PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ – Read Output 180
Table 125: PROFIBUS_FSPMM2_CMD_READ_OUTPUT_CNF – Confirmation of Read Output 181
Table 126: PROFIBUS_FSPMM2_CMD_GET_CFG_REQ – Get Configuration 182
Table 127: PROFIBUS_FSPMM2_CMD_GET_CFG_CNF – Confirmation of Get Configuration 183
Table 128: PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ – Set Slave Address 185
Table 129: PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_CNF – Confirmation of Set Slave Address 186
Table 130: PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ – Get Master Diagnosis 187
Table 131: PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_CNF – Confirmation of Get Master Diagnosis 188
Table 132: Coding of Live List Bytes 189
Table 133: PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ - – Live List Command 190
Table 134: PROFIBUS_FSPMM2_CMD_LIVE_LIST_CNF – Confirmation of Live List Request 191
Table 135: PROFIBUS_FSPMM2_CMD_ABORT_IND - Abort Indication 193
Table 136: PROFIBUS_FSPMM2_CMD_ABORT_RES – Response to Abort Indication 194
Table 137: PROFIBUS_FSPMM2_CMD_CLOSED_IND - Closed Indication 195
Table 138: PROFIBUS_FSPMM2_CMD_CLOSED_RES –Response to Closed Indication 196
Table 139: PROFIBUS_FSPMM2_CMD_RESET_CONN_REQ – Reset Connection Request 197
Table 140: PROFIBUS_FSPMM2_CMD_RESET_CONN_CNF – Confirmation of Reset Connection Request 198
Table 141: FSPMM-Task process queue 199
Table 142: Overview over the Packets of the PROFIBUS DL -Task of the PROFIBUS DP-Master Protocol Stack 200
Table 143: PROFIBUS_DL_CMD_START_DLE_REQ – Start the DL-Layer Request 200
Table 144: PROFIBUS_DL_CMD_START_DLE_CNF – Confirmation of Start the DL-Layer Request 201
Table 145: PROFIBUS_DL_CMD_STOP_DLE_REQ– Stop DL Layer Request 202
Table 146: PROFIBUS_DL_CMD_STOP_DLE_CNF – Confirmation of Stop DL Layer Request 203
Table 147: PROFIBUS_DL_CMD_DATA_ACK_REQ - Send SDA Service 205
Table 148: PROFIBUS_DL_CMD_DATA_ACK_CNF – Confirmation of Send SDA Service 206
Table 149: PROFIBUS_DL_CMD_DATA_ACK_CNF - Packet Status/Error 207
Table 150: Reported Errors, their Sources and Possible Actions 208
Table 151: PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service 210
Table 152: PROFIBUS_DL_CMD_DATA_REQ - Send SDN Service Request 212
Table 153: PROFIBUS_DL_CMD_DATA_CNF – Confirmation of Send SDN Service Request 213
Table 154: PROFIBUS_DL_CMD_DATA_CNF - Packet Status/Error 214
Table 155: Reported Errors, their Sources and Possible Actions 214
Table 156: PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication 216
Table 157: PROFIBUS_DL_CMD_DATA_REPLY_REQ - Send SRD Service Request 218
Table 158: PROFIBUS_DL_CMD_DATA_REPLY_CNF – Confirmation of Send SRD Service Request 219
Table 159: PROFIBUS_DL_CMD_DATA_REPLY_CNF - Packet Status/Error 220
Table 160: Reported Errors, their Sources and Possible Actions 220
Table 161: PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication 222
Table 162: Allowed Values and their Meanings for Variable bUpdateStatus 223
Table 163: PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ - Reply Update Service 225
Table 164: PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_CNF – Confirmation of Reply Update Service 227
Table 165: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ - Activate a SAP for Request 229
Table 166: Values of bServiceActivate Parameter and the according Service 230
Table 167: Values of bRoleInService Parameter and the according Operation 230
Table 168: Allowed combinations of bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh parameters for services SDA and SDN 231
Table 169: Allowed combinations of bMaxDLSDULenReqLow, bMaxDLSDULenReqHigh, bMaxDLSDULenIndCnfLow and bMaxDLSDULenIndCnfHigh parameters for services SRD and MSRD 231
Table 170: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_CNF - Confirmation of Activate a SAP for Request 232
Table 171: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ - Activate a SAP for Responder Functionality 234
Table 172: Allowed combinations of length parameters for service SRD with role in service ‘Responder’ 235
Table 173: Allowed Values and their Meanings for Variable bUpdateStatus 236

Table 174: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_CNF – Confirmation of Activate a SAP for Responder Functionality 237
Table 175: PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ - Deactivate an SAP for Request 238
Table 176: PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_CNF – Confirmation of Deactivate an SAP for Request 239
Table 177: PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ - Load the Bus Parameter Set 242
Table 178: PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_CNF – Confirmation of Loading the Bus Parameter Set 243
Table 179: PROFIBUS_DL_CMD_SET_VALUE_REQ - Set Value Service 245
Table 180: PROFIBUS_DL_CMD_SET_VALUE_CNF – Confirmation of Set Value Service 246
Table 181: PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ – Obtain FDL Status Request 247
Table 182: PROFIBUS_DL_CMD_SEND_FDL_STATUS_CNF – Confirmation of Obtain FDL Status Request 248
Table 183: PROFIBUS_DL_CMD_GET_LMS_REQ - Get List of active Stations 250
Table 184: PROFIBUS_DL_CMD_GET_LMS_CNF - – Confirmation of Get List of active Stations Request 251
Table 185: PROFIBUS_DL_CMD_REGISTER_LMS_REQ - Register an Application to receive LMS Change Indications 253
Table 186: PROFIBUS_DL_CMD_REGISTER_LMS_CNF – Confirmation of Register an Application to receive LMS Change Indications Request 254
Table 187: PROFIBUS_DL_CMD_LMS_CHANGED_IND - Indication for acknowledged connectionless Data Transfer 256
Table 188: Possible Values of Single Bytes in Response 257
Table 189: PROFIBUS_DL_CMD_GET_LL_REQ - Get Life List Request 258
Table 190: PROFIBUS_DL_CMD_GET_LL_CNF - Confirmation to Get Life List Request 259
Table 191: FSPMM-Task process queue 260
Table 192: Overview over the Packets of the APM-Task of the PROFIBUS DP Protocol Stack 260
Table 193: PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ 262
Table 194: PROFIBUS_APM_CMD_REDUNDANT_MODE_CNF 263
Table 195: PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master 264
Table 196: Overview over Packets required for Configuration “on the run” 265
Table 197: RCX_VERIFY_DATABASE_REQ – Verify the new Configuration Database 266
Table 198: RCX_VERIFY_DATABASE_CNF – Confirmation of Verify the new Configuration Database 268
Table 199: RCX_ACTIVATE_DATABASE_REQ – Activate the new Configuration Database 269
Table 200: RCX_ACTIVATE_DATABASE_CNF - Confirmation of Activate the new Configuration Database 270
Table 201: Supported modes of PROFIBUS-DP Master protocol stack 271
Table 202: Handshake Parameters for PROFIBUS DP Master 274
Table 203: Names of Queues in PROFIBUS DP Master Firmware 276
Table 204: Common Error codes 279
Table 205: Error Codes of the FSPMM-Task 281
Table 206: Diagnostic Codes of the FSPMM-Task 282
Table 207: Error Codes of the FSPMM-Task 283
Table 208: Error Codes of the DL-Task 285
Table 209: Diagnostic Codes of the DL-Task 285
Table 210: Error Codes of the APM-Task 286
Table 211: Diagnostic Codes of the APM-Task 287

9.4 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com
Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn
Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr
Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it
Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp
Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch
Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us
Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com