



**Protocol API**  
**PROFIBUS DP-Master**

V2.3.x.x

**Hilscher Gesellschaft für Systemautomation mbH**

**[www.hilscher.com](http://www.hilscher.com)**

DOC061001API12EN | Revision 12 | English | 2010-12 | Released | Public

## Revision History

Rev	Date	Name	Revisions
1	2006-10-09	RH	- Create first draft
2	2006-11-03	MK	- Add. Command PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF - Replaced all undefined struct elements "TLR_UINT" with "TLR_UINT32"
3	2007-02-28	RG	- Added a lot of material to the Profibus DP Master Programming Interface Manual.
4	2007-08-21	RG/RH	- Added a new chapter "Overview". - Added a lot of additional material - Added the descriptions of master status block, bus and slave parameters. - Description of new error messages added. - Correction of some minor faults.
5	2007-12-14	RG/RH	- PROFIBUS-DP Master firmware version 2.0.
6	2008-06-02	RG MK	- Review of technical data section - Correction in chapter 6.2.2; 6.2.5; 6.3.12; 6.3.13, 6.3.4 to 6.3.9 Firmware/stack version 2.0.32. Reference to netX Dual-Port Memory Interface Manual Revision 5.
7	2008-12-03	RG/UJ	- Firmware / Stack version 2.2.2 - Reference to netX Dual-Port Memory Interface Manual Revision 7 - Changed some error numbers to global error numbers - Some changes in allowed ranges of values, small corrections. - New Function "Activate/Deactivate Slaves" chapter 6.2.15 - New parameter in "Master_User_Data" of the bus parameter set - explanation of the Clear_Data bit in the global control command
8	2009-04-18	RG/ET	- Firmware / Stack version 2.2.4 - Added description of PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ/CNF – Obtain FDL Status packet of PROFIBUS_DL task. - Additionally, a lot of smaller corrections were made.
9	2009-06-03	RG/RH	- Firmware / Stack version 2.2.5 - Added description of <i>Redundancy Functionality</i> (based on SYCON.net configuration) - Added packet description for redundancy functionality - Figure 6: Internal Structure of PROFIBUS DP-Master Firmware updated
10	2009-11-11	RG/RH/HH	- Firmware / Stack version 2.2.13 - Added description of configuration on the run (during OPERATE state) - Error correction concerning <i>Identification-related diagnosis</i> - Extended section on task structure with descriptions of single tasks. - Added information of suitability of packets for LFW or LOM approach.

Rev	Date	Name	Revisions
11	2010-05-12	RG/RH	<ul style="list-style-type: none"> <li>- Firmware / Stack version 2.3.x.x</li> <li>- Extended Status section adapted to new standard</li> <li>- Added device status</li> </ul> <p>Section <i>Technical Data</i>: New: Support of DMA for PCI targets and support of slot number for CIFX 50-DP</p> <ul style="list-style-type: none"> <li>- Reference to netX Dual-Port Memory Interface Manual Revision 9.</li> </ul>
12	2010-11-22	RG	<ul style="list-style-type: none"> <li>- Firmware / Stack version 2.3.22.x</li> <li>- Reference to netX Dual-Port Memory Interface Manual Revision 9.</li> <li>- Added new section "<i>Obtaining Diagnostic Information from connected Slaves by sending an RCX_GET_SLAVE_CONN_INFO_REQ Packet</i>"</li> </ul>

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>12</b>
1.1	Abstract	12
1.2	Functional Overview	12
1.3	System Requirements	12
1.4	Intended Audience	12
1.5	Specifications	13
1.5.1	Technical Data	13
1.6	Terms, Abbreviations and Definitions	15
1.7	References	15
1.8	Legal Notes	16
1.8.1	Copyright	16
1.8.2	Important Notes	16
1.8.3	Exclusion of Liability	17
1.8.4	Export	17
<b>2</b>	<b>Fundamentals</b>	<b>18</b>
2.1	General Access Mechanisms on netX Systems	18
2.2	Accessing the Protocol Stack by Programming the AP Task's Queue	19
2.2.1	Getting the Receiver Task Handle of the Process Queue	19
2.2.2	Meaning of Source- and Destination-related Parameters	19
2.3	Accessing the Protocol Stack via the Dual Port Memory Interface	20
2.3.1	Communication via Mailboxes	20
2.3.2	Using Source and Destination Variables correctly	21
2.3.3	Obtaining useful Information about the Communication Channel	24
2.4	Client/Server Mechanism	26
2.4.1	Application as Client	26
2.4.2	Application as Server	27
<b>3</b>	<b>Dual-Port-Memory</b>	<b>28</b>
3.1	Cyclic Data (Input/Output Data)	28
3.1.1	Input Data Image	29
3.1.2	Process Data Output	29
3.2	Acyclic Data (Mailboxes)	30
3.2.1	General Structure of Messages or Packets for Non-Cyclic Data Exchange	31
3.2.2	Status & Error Codes	33
3.2.3	Differences between System and Channel Mailboxes	33
3.2.4	Send Mailbox	33
3.2.5	Receive Mailbox	34
3.2.6	Channel Mailboxes (Details of Send and Receive Mailboxes)	34
3.3	Status	35
3.3.1	Common Status	35
3.3.2	Extended Status	43
3.4	Control Block	53
<b>4</b>	<b>Getting started / Configuration</b>	<b>54</b>
4.1	Overview about Essential Functionality	54
4.2	Configuration of Bus and Slave Parameters	55
4.2.1	Write Access to the Dual-Port Memory	55
4.2.2	Using the configuration tool SYCON.net	55
4.2.3	Detailed Description of Bus and Master Parameters	56
4.2.4	Detailed Description of Slave Parameters	62
4.3	Task Structure of the PROFIBUS DP-Master Stack	69
4.3.1	Obtaining Diagnostic Information from connected Slaves by sending an RCX_GET_SLAVE_CONN_INFO_REQ Packet	71
<b>5</b>	<b>Overview</b>	<b>73</b>
5.1	Profibus DP and the OSI/ISO Layer Model	73
5.2	Profibus DP Operation Modes (States)	74
5.3	Functionality of the FSPMM-Task (Layer 7)	75
5.3.1	Cyclic Data Transfer	75
5.3.2	Acyclic Data Transfer	77

5.3.3	Configuration of Inputs and Outputs at Slaves .....	79
5.3.4	Alarm Processing .....	82
5.3.5	Diagnosis .....	84
5.4	Functionality of the FSPMM2-Task .....	92
5.4.1	General Remarks on DP V1 Class 2 .....	92
5.4.2	DP V1 Class 2 Masters .....	92
5.4.3	Basic Services for Connection Maintenance .....	92
5.4.4	Basic Services available after Establishing a DP V1-Class 2 Connection .....	93
5.4.5	Overview of Supported Functionality .....	93
5.4.6	Extended Addressing Mechanism .....	94
5.4.7	Short Description of the MSAC2M State Machine .....	94
5.4.8	Initialization Process of a DP V1-Class 2 Connection .....	95
5.4.9	Important Parameters .....	96
5.4.10	Requirements for the Class 2 Master's User Task .....	97
5.4.11	Connection Supervision by Internal Timers .....	97
5.5	Functionality of the DL Task (Layer 2) .....	98
5.5.1	Data Transfer Services .....	99
5.5.2	Management of Services Access Points .....	102
5.5.3	Management of Layer 2 System Variables and Tasks .....	103
5.6	Configuration during running system (Operation State OPERATE) .....	104
5.6.1	Configuration Procedure .....	104
5.7	Redundancy Functionality .....	106
<b>6</b>	<b>The Application Interface .....</b>	<b>108</b>
6.1	The FSPMM-Task .....	109
6.1.1	PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register .....	111
6.1.2	PROFIBUS_FSPMM_CMD_INIT_REQ/CNF – Initialization Command .....	113
6.1.3	PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode .....	116
6.1.4	PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication .....	119
6.1.5	PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic .....	121
6.1.6	PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic .....	126
6.1.7	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output Data .....	127
6.1.8	PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input Data .....	130
6.1.9	PROFIBUS_FSPMM_CMD_READ_REQ/CNF –V1 Class 1 Read Request .....	133
6.1.10	PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – V1 Class 1 Write Request .....	136
6.1.11	PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification .....	141
6.1.12	PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge .....	144
6.1.13	PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ/CNF – Download Slave Parameter Set .....	148
6.1.14	PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF – Global Control Message .....	155
6.1.15	PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ/CNF – Activate/Deactivate Slaves .....	160
6.1.16	PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a Fatal Fault .....	163
6.2	The FSPMM2-Task .....	165
6.2.1	PROFIBUS_FSPMM2_CMD_INIT_REQ/CNF – Initialization Command .....	167
6.2.2	PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate DPV1C2 Connection .....	170
6.2.3	PROFIBUS_FSPMM2_CMD_READ_REQ/CNF - V1 Class 2 Read Request .....	179
6.2.4	PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class 2 Write Request .....	184
6.2.5	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined V1 Class 2 Read and Write Request .....	191
6.2.6	PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection .....	198
6.2.7	PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ/CNF – Read Slave Diagnostics (DP V1 Class 2) .....	203
6.2.8	PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ/CNF – Read Input Values .....	206
6.2.9	PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ/CNF – Read Output Values .....	209
6.2.10	PROFIBUS_FSPMM2_CMD_GET_CFG_REQ/CNF – Get Configuration Command .....	212
6.2.11	PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ/CNF – Set Slave Address (DP V1 Class 2) .....	215
6.2.12	PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ/CNF – Get Master Diagnosis .....	218
6.2.13	PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ/CNF – Live List Command .....	221
6.2.14	PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication .....	224
6.2.15	PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication .....	228
6.3	The DL-Task .....	231
6.3.1	PROFIBUS_DL_CMD_END_PROCESS_REQ/CNF– Finish the DL Task .....	233
6.3.2	PROFIBUS_DL_CMD_START_DLE_REQ/CNF – Starts the DL-Layer .....	235
6.3.3	PROFIBUS_DL_CMD_STOP_DLE_REQ/CNF – Stop DL Layer .....	237
6.3.4	PROFIBUS_DL_CMD_DATA_ACK_REQ/CNF - Send SDA Service .....	239
6.3.5	PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service .....	245
6.3.6	PROFIBUS_DL_CMD_DATA_REQ/CNF - Send SDN Service .....	248

6.3.7	PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication.....	254
6.3.8	PROFIBUS_DL_CMD_DATA_REPLY_REQ/CNF - Send SRD Service.....	257
6.3.9	PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication.....	264
6.3.10	PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ/CNF - Reply Update Service.....	267
6.3.11	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request.....	271
6.3.12	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ/CNF - Activate an SAP for Responder Functionality.....	276
6.3.13	PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request	281
6.3.14	PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set.....	284
6.3.15	PROFIBUS_DL_CMD_SET_VALUE_REQ/CNF - Set Value Service.....	290
6.3.16	PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ/CNF - Obtain FDL Status.....	293
6.4	The APM-Task.....	295
6.4.1	PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ/CNF - Set Master to active/passive.....	296
6.4.2	PROFIBUS_APM_CMD_REDUNDANT_MODE_IND - Indicate Status changed of active Master	299
6.5	Packets for Configuration during running system.....	300
6.5.1	RCX_VERIFY_DATABASE_REQ/CNF - Verify the new Configuration Database.....	301
6.5.2	RCX_ACTIVATE_DATABASE_REQ/CNF - Activate the new Configuration Database.....	304
<b>7</b>	<b>Status/Error codes overview.....</b>	<b>306</b>
7.1	Error Codes of the FSPMM-Task.....	306
7.1.1	Diagnostic Codes of the FSPMM-Task.....	308
7.2	Error Codes of the FSPMM2-Task.....	309
7.2.1	Diagnostic Codes of the FSPMM2-Task.....	309
7.3	Error Codes of the DL-Task.....	310
7.3.1	Diagnostic Codes of the DL-Task.....	312
7.4	Error Codes of the APM-Task.....	313
7.4.1	Diagnostic Codes of the APM-Task.....	314
<b>8</b>	<b>Contact.....</b>	<b>315</b>

---

## List of Figures

Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System.....	18
Figure 2: Use of <code>ulDest</code> in Channel and System Mailbox .....	21
Figure 3: Using <code>ulSrc</code> and <code>ulSrcId</code> .....	22
Figure 4: Transition Chart Application as Client.....	26
Figure 5: Transition Chart Application as Server.....	27
Figure 6: Internal Structure of PROFIBUS DP-Master Firmware .....	69
Figure 7: Profibus in the OSI/ISO Layer Model.....	73
Figure 8: Initialization Sequence of DP V1 Class 2-Connection .....	95
Figure 9: SDA Service with Acknowledgement .....	99
Figure 10: Service Access Points .....	102
Figure 11: Configuration during network state OPERATE .....	105
Figure 12: Redundancy Functionality – Host Application.....	107

## List of Tables

Table 1: Terms, Abbreviations and Definitions .....	15
Table 2: References.....	15
Table 3: Documents published by PROFIBUS International.....	15
Table 4: Input Data Image .....	29
Table 5: Output Data Image.....	29
Table 6: Channel Mailboxes .....	34
Table 7: Common Status Structure Definition.....	36
Table 8: Communication State of Change .....	37
Table 9: Meaning of Communication Change of State Flags.....	38
Table 10: Master Status Structure Definition .....	41
Table 11: rcX Status and Error Codes/Slave .....	42
Table 12: Extended Status Block.....	43
Table 13: Extended Status Block for PROFIBUS DP-Master – First part .....	45
Table 14: Contents of Global Bits Variable .....	45
Table 15: Error Types in Global Bits.....	46
Table 16: Operation Modes of the Profibus DP Master and their Values.....	46
Table 17: Relationship between Slave Station Address and the corresponding abSl_cfg Bit.....	47
Table 18: Relationship between Slave Station Address and the corresponding abSl_state Bit.....	48
Table 19: Relationship between Slave Station Address and the corresponding abSl_diag Bit.....	48
Table 20: Relationship between abSl_state and abSl_diag bits .....	49
Table 21: Errors which may occur in the Network (bErr_Rem_Adr is not equal to 255) .....	49
Table 22: Errors which may occur in the Profibus DP Master Device (bErr_Rem_Adr is equal to 255).....	51
Table 23: Extended Status Block for PROFIBUS DP-Master – Second part (State Field Definition Block) .....	52
Table 24: Overview about Essential Functionality (Cyclic and acyclic Data Transfer and Alarm Handling).....	54
Table 25: Bus and Master Parameters, their Meanings and their Ranges of allowed Values.....	56
Table 26: Supported Baud Rates.....	57
Table 27: Slave Parameters, their Meanings and their Ranges of allowed Values.....	62
Table 28: Explanation of Bits of Sl_Flag within Slave Parameter Block.....	63
Table 29: Meaning of Slave_Type .....	64
Table 30: Meaning of Alarm_Mode Parameter .....	64
Table 31: Explanation of Bits of Sl_Flag within Slave Parameter Block.....	65
Table 32: Octet 1 of Prm_Data Slave Parameter.....	66
Table 33: Meaning of Combinations of Lock_Req and Unlock_Req Bits .....	66
Table 34: abAddTab Structure .....	68
Table 35: Packets for Cyclic Data Transfer.....	75
Table 36: Packets for Acyclic Data Transfer.....	77
Table 37: Explanation of Error Class and Error Code within Error_Code_1 .....	78
Table 38: General Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification .....	80
Table 39: Special Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification .....	81
Table 40: Structure of Length Byte in the Special Identifier Format of the Identifier Byte according to IEC 61158/ EN 50170 Specification .....	81
Table 41: Octet 1: Station Status_1 .....	85
Table 42: Octet 2: Station Status_2 .....	86
Table 43: Octet 3: Station Status_3 .....	87
Table 44: Header of Device-related Diagnosis .....	88
Table 45: Header of Identification-related Diagnosis .....	88
Table 46: Identification-related diagnosis – Second Byte .....	89
Table 47: Identification-related diagnosis – Third Byte .....	89
Table 48: Header of Channel-related Diagnosis .....	90
Table 49: Channel-related Diagnosis – Second Byte.....	90
Table 50: Channel-related Diagnosis – Third Byte.....	91
Table 51: Overview of available Packets supporting DP V1 Class 2 .....	93
Table 52: FSPMM-Task process queue.....	109
Table 53: Table 25: Overview over the Packets of the FSPMM-Master -Task of the PROFIBUS DP-Master Protocol Stack .....	110
Table 54: PROFIBUS_FSPMM_APP_REG_REQ – Application Register Request .....	111
Table 55: PROFIBUS_FSPMM_APP_REG_REQ– Packet Status/Error .....	111
Table 56: PROFIBUS_FSPMM_APP_REG_CNF – Application Register Confirmation.....	112
Table 57: PROFIBUS_FSPMM_APP_REG_CNF - Packet Status/Error .....	112
Table 58: PROFIBUS_FSPMM_CMD_INIT_REQ – Request for setting of Bus Parameters.....	114
Table 59: PROFIBUS_FSPMM_CMD_INIT_CNF - Confirmation Command for Request for Setting of Bus Parameter.....	115
Table 60: PROFIBUS_FSPMM_CMD_SET_MODE_REQ– Set a new operation mode.....	117
Table 61: PROFIBUS_FSPMM_CMD_SET_MODE_/CNF – Set a new operation mode .....	118

Table 62: PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication.....	120
Table 63: ulFlags Bit Mask.....	122
Table 64: PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ – Request a Slave Diagnostic .....	123
Table 65: PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_CNF – Confirmation of Request a Slave Diagnostic .....	125
Table 66: PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic .....	126
Table 67: PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ– Set new Output Data .....	128
Table 68: PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF – Confirmation of Setting new Output Data.....	129
Table 69: PROFIBUS_FSPMM_CMD_SET_OUTPUT_CNF – Packet Status/Error.....	129
Table 70: PROFIBUS_FSPMM_CMD_GET_INPUT_REQ - Get new Input Data.....	130
Table 71: PROFIBUS_FSPMM_CMD_GET_INPUT_CNF - Confirmation of Get new Input Data .....	131
Table 72: PROFIBUS_FSPMM_CMD_READ_REQ –V1 Class 1 Read Request .....	134
Table 73: PROFIBUS_FSPMM_CMD_READ_CNF –Confirmation of V1 Class 1 Read Request .....	135
Table 74: PROFIBUS_FSPMM_CMD_WRITE_REQ– V1 Class 1 Write Request.....	137
Table 75: PROFIBUS_FSPMM_CMD_WRITE_CNF – Confirmation of V1 Class 1 Write Request.....	139
Table 76: PROFIBUS_FSPMM_CMD_WRITE_CNF – Packet Status/Error.....	140
Table 77: Available Alarm Types .....	141
Table 78: Flags of Alarm_Spec_Ack parameter .....	142
Table 79: Meaning of Alarm_Specifier Bits D1 and D0.....	142
Table 80: PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification .....	143
Table 81: PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ – Alarm Acknowledge.....	145
Table 82: PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF – Confirmation of Alarm Acknowledge.....	146
Table 83: Possible Errors and Descriptions .....	147
Table 84: PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ– Download Slave Parameter Set .....	149
Table 85: Structure of the Slave Parameter Set according to IEC 61158 Specification .....	150
Table 86: abAddTab Structure .....	151
Table 87: Example of a Slave Parameter Data Set.....	153
Table 88: PROFIBUS_FSPMM_CMD_DOWNLOAD_CNF – Configuration of Download Slave Parameter Set .	154
Table 89: Explanation of Bits in Global Control_Command .....	155
Table 90: Explanations of possible Combinations of the Bits 'Unsync'/ 'Sync' and 'Unfreeze'/ 'Freeze':.....	156
Table 91: PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ– Send a Global Control Request.....	158
Table 92: PROFIBUS_FSPMM_CMD_GLOBALCONTROL_CNF – Confirmation of Sending a Global Control Message.....	159
Table 93: PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ – Slave Activate Request Packet .....	161
Table 94: PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_CNF – Slave Activate Confirmation Packet.....	162
Table 95: PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a fatal Fault .....	164
Table 96: FSPMM-Task process queue.....	165
Table 97: Table 25: Overview over the Packets of the FSPMM-Master -Task of the PROFIBUS DP-Master Protocol Stack .....	166
Table 98: PROFIBUS_FSPMM2_CMD_INIT_REQ- Initialization Command for Class 2 Connection.....	168
Table 99: PROFIBUS_FSPMM2_CMD_INIT_CNF - Confirmation of Initialization Command .....	169
Table 100: PROFIBUS_FSPMM2_CMD_INITIATE_REQ - Initiate Command.....	174
Table 101: PROFIBUS_FSPMM2_CMD_INITIATE_CNF_POS - Positive Confirmation of Initiate Command ...	177
Table 102: PROFIBUS_FSPMM2_CMD_INITIATE_CNF_NEG - Negative Confirmation of Initiate Command ..	178
Table 103: PROFIBUS_FSPMM2_CMD_READ_REQ – Read Command .....	180
Table 104: PROFIBUS_FSPMM2_CMD_READ_CNF_POS - Positive Confirmation of Read Command .....	182
Table 105: PROFIBUS_FSPMM2_CMD_READ_CNF_POS - Packet Status/ErrorPacket Description .....	182
Table 106: PROFIBUS_FSPMM2_CMD_READ_CNF_NEG - Negative Confirmation of Read Command .....	183
Table 107: PROFIBUS_FSPMM2_CMD_WRITE_REQ – Write Command.....	186
Table 108: PROFIBUS_FSPMM2_CMD_WRITE_CNF_POS - Positive Confirmation of Write Command.....	188
Table 109: PROFIBUS_FSPMM2_CMD_WRITE_CNF_NEG - Negative Confirmation of Write Command.....	190
Table 110: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ – Data Transport Command .....	193
Table 111: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF_POS - Positive Confirmation of Data Transport Command .....	195
Table 112: PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_CNF_NEG - Negative Confirmation of Data Transport Command .....	197
Table 113: Allowed Values of Subnet Variable .....	198
Table 114: Instance Codes and their Meaning.....	198
Table 115: Possible Reason Codes and their Meaning .....	199
Table 116: PROFIBUS_FSPMM2_CMD_ABORT_REQ – Abort Request.....	201
Table 117: PROFIBUS_FSPMM2_CMD_ABORT_CNF - Confirmation of Abort Request.....	202
Table 118: PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ - Read Slave Diagnostics (DP V1 Class 2) .....	204
Table 119: PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_CNF - Confirmation of Read Slave Diagnostics (DP V1 Class 2).....	205
Table 120: PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ – Read Input Request.....	207
Table 121: PROFIBUS_FSPMM2_CMD_READ_INPUT_CNF – Confirmation of Read Input .....	208

Table 122: PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ – Read Output .....	210
Table 123: PROFIBUS_FSPMM2_CMD_READ_OUTPUT_CNF – Confirmation of Read Output .....	211
Table 124: PROFIBUS_FSPMM2_CMD_GET_CFG_REQ – Get Configuration .....	213
Table 125: PROFIBUS_FSPMM2_CMD_GET_CFG_CNF – Confirmation of Get Configuration .....	214
Table 126: PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ – Set Slave Address.....	216
Table 127: PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_CNF – Confirmation of Set Slave Address.....	217
Table 128: PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ – Get Master Diagnosis.....	219
Table 129: PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_CNF – Confirmation of Get Master Diagnosis	220
Table 130: Coding of Live List Bytes.....	221
Table 131: PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ - - Live List Command.....	222
Table 132: PROFIBUS_FSPMM2_CMD_LIVE_LIST_CNF – Confirmation of Live List Request .....	223
Table 133: PROFIBUS_FSPMM2_CMD_ABORT_IND - Abort Indication .....	226
Table 134: PROFIBUS_FSPMM2_CMD_ABORT_RES – Response to Abort Indication .....	227
Table 135: PROFIBUS_FSPMM2_CMD_CLOSED_IND - Closed Indication .....	229
Table 136: PROFIBUS_FSPMM2_CMD_CLOSED_RES –Response to Closed Indication .....	230
Table 137: FSPMM-Task process queue.....	231
Table 138: Overview over the Packets of the PROFIBUS_DL -Task of the PROFIBUS DP-Master Protocol Stack .....	232
Table 139: PROFIBUS_DL_CMD_END_PROCESS_REQ– Finish the DL Task.....	233
Table 140: PROFIBUS_DL_CMD_END_PROCESS_CNF– Finish the DL Task.....	234
Table 141: PROFIBUS_DL_CMD_START_DLE_REQ – Start the DL-Layer Request .....	235
Table 142: PROFIBUS_DL_CMD_START_DLE_CNF – Confirmation of Start the DL-Layer Request .....	236
Table 143: PROFIBUS_DL_CMD_STOP_DLE_REQ– Stop DL Layer Request .....	237
Table 144: PROFIBUS_DL_CMD_STOP_DLE_CNF – Confirmation of Stop DL Layer Request.....	238
Table 145: PROFIBUS_DL_CMD_DATA_ACK_REQ - Send SDA Service.....	240
Table 146: PROFIBUS_DL_CMD_DATA_ACK_CNF – Confirmation of Send SDA Service .....	242
Table 147: PROFIBUS_DL_CMD_DATA_ACK_CNF - Packet Status/Error.....	243
Table 148: Reported Errors, their Sources and Possible Actions .....	244
Table 149: PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service.....	246
Table 150: PROFIBUS_DL_CMD_DATA_REQ - Send SDN Service Request .....	249
Table 151: PROFIBUS_DL_CMD_DATA_CNF – Confirmation of Send SDN Service Request.....	251
Table 152: PROFIBUS_DL_CMD_DATA_CNF - Packet Status/Error .....	252
Table 153: Reported Errors, their Sources and Possible Actions .....	253
Table 154: PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication.....	255
Table 155: PROFIBUS_DL_CMD_DATA_REPLY_REQ - Send SRD Service Request.....	258
Table 156: PROFIBUS_DL_CMD_DATA_REPLY_CNF – Confirmation of Send SRD Service Request .....	261
Table 157: PROFIBUS_DL_CMD_DATA_REPLY_CNF - Packet Status/Error .....	262
Table 158: Reported Errors, their Sources and Possible Actions .....	263
Table 159: PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication.....	265
Table 160: Allowed Values and their Meanings for Variable bUpdateStatus .....	266
Table 161: PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ - Reply Update Service.....	268
Table 162: PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_CNF – Confirmation of Reply Update Service .....	270
Table 163: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ - Activate a SAP for Request.....	272
Table 164: Values of bServiceActivate Parameter and the according Service.....	273
Table 165: Values of bRoleInService Parameter and the according Operation .....	273
Table 166: Allowed combinations of bMaxDLSDUlenReqLow, bMaxDLSDUlenReqHigh, bMaxDLSDUlenIndCnfLow and bMaxDLSDUlenIndCnfHigh parameters for services SDA and SDN.....	274
Table 167: Allowed combinations of bMaxDLSDUlenReqLow, bMaxDLSDUlenReqHigh, bMaxDLSDUlenIndCnfLow and bMaxDLSDUlenIndCnfHigh parameters for services SRD and MSRD .....	274
Table 168: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_CNF - Confirmation of Activate a SAP for Request.....	275
Table 169: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ - Activate a SAP for Responder Functionality .....	277
Table 170: Allowed combinations of bMaxDLSDUlenReqLow, bMaxDLSDUlenReqHigh, bMaxDLSDUlenIndLow and bMaxDLSDUlenIndHigh parameters for service SRD with role in service 'Responder' .....	278
Table 171: Allowed Values and their Meanings for Variable bUpdateStatus .....	279
Table 172: PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_CNF – Confirmation of Activate a SAP for Responder Functionality.....	280
Table 173: PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ - Deactivate an SAP for Request.....	282
Table 174: PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_CNF – Confirmation of Deactivate an SAP for Request .....	283
Table 175: PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ - Load the Bus Parameter Set .....	286
Table 176: PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_CNF – Confirmation of Loading the Bus Parameter Set.....	289
Table 177: PROFIBUS_DL_CMD_SET_VALUE_REQ - Set Value Service .....	291

---

Table 178: PROFIBUS_DL_CMD_SET_VALUE_CNF – Confirmation of Set Value Service .....	292
Table 179: PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ – Obtain FDL Status Request.....	293
Table 180: PROFIBUS_DL_CMD_SEND_FDL_STATUS_CNF – Confirmation of Obtain FDL Status Request ..	294
Table 181: FSPMM-Task process queue.....	295
Table 182: Overview over the Packets of the APM-Task of the PROFIBUS DP Protocol Stack.....	295
Table 183: PROFIBUS_APM_CMD_REDUNDANT_MODE_REQ .....	297
Table 184: PROFIBUS_APM_CMD_REDUNDANT_MODE_CNF .....	298
Table 185: PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master ...	299
Table 186: Overview over Packets required for Configuration “on the run” .....	300
Table 187: RCX_VERIFY_DATABASE_REQ – Verify the new Configuration Database.....	301
Table 188: RCX_VERIFY_DATABASE_CNF – Confirmation of Verify the new Configuration Database.....	303
Table 189: RCX_ACTIVATE_DATABASE_REQ – Activate the new Configuration Database .....	304
Table 190: RCX_ACTIVATE_DATABASE_CNF - Confirmation of Activate the new Configuration Database ....	305
Table 191: Error Messages of the FSPMM-Task.....	307
Table 192: Diagnostic Messages of the FSPMM-Task .....	308
Table 193: Error Messages of the FSPMM-Task.....	309
Table 194: Error Messages of the DL-Task .....	311
Table 195: Diagnostic Messages of the DL-Task.....	312
Table 196: Error Messages of the APM-Task .....	313
Table 197: Diagnostic Messages of the APM-Task .....	314

# 1 Introduction

## 1.1 Abstract

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 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 Class 1
- Baud rate ranging from 9.6 kBaud up to 12Mbaud

## 1.3 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.

## 1.4 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

## 1.5 Specifications

The data below applies to PROFIBUS-DP Master firmware and stack version 2.3.x.x.

### 1.5.1 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 Master acyclic class 1 state machine

#### Technical Data

Maximum number of supported DPV0/DPV1 slaves	125
Maximum number of total cyclic input data	3584 bytes (status information is separately managed)
Maximum number of total cyclic output data	3584 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	DP V1 Class 1 Read/Write DP V1 Class 1 Alarm DP V1 Class 2 Initiate/Read/Write/Abort
Maximum number of acyclic read/write	240 bytes/slave
Baud rate	Fixed values from 9,6 kBits/s to 12 MBit/s Auto-detection mode is not supported.
Data transport layer	PROFIBUS FDL
Redundancy function	supported (Requires host application support)

#### Firmware/stack available for netX

netX 50	no
netX 100, netX 500	yes



## 1.6 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	Field bus Service Protocol Machine Master
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 1: 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.

## 1.7 References

This document is based on the following specifications:

1	DPM Interface Manual for netX based Products, Revision 7; Hilscher Gesellschaft für Systemautomation mbH; 2007-2009
2	IEC 61158 Third edition, 2003

Table 2: References

There are also some documents available from the users' organization Profibus International, which contain all relevant parts of the Profibus DP specifications and can be used alternatively to reference #3. These are:

1	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
2	Profibus document #2.082: Technical Guideline PROFIBUS-DP Extensions to EN 50170 - Version 2.0, April 1998

Table 3: Documents published by PROFIBUS International

## **1.8 Legal Notes**

### **1.8.1 Copyright**

© 2006-2010 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying material (user manual, accompanying texts, documentation, etc.) are protected by German and international copyright law as well as international trade and protection provisions. You are not authorized to duplicate these in whole or in part using technical or mechanical methods (printing, photocopying or other methods), to manipulate or transfer using electronic systems without prior written consent. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. The included diagrams do not take the patent situation into account. The company names and product descriptions included in this document may be trademarks or brands of the respective owners and may be trademarked or patented. Any form of further use requires the explicit consent of the respective rights owner.

### **1.8.2 Important Notes**

The user manual, accompanying texts and the documentation were created for the use of the products by qualified experts, however, errors cannot be ruled out. For this reason, no guarantee can be made and neither juristic responsibility for erroneous information nor any liability can be assumed. Descriptions, accompanying texts and documentation included in the user manual do not present a guarantee nor any information about proper use as stipulated in the contract or a warranted feature. It cannot be ruled out that the user manual, the accompanying texts and the documentation do not correspond exactly to the described features, standards or other data of the delivered product. No warranty or guarantee regarding the correctness or accuracy of the information is assumed.

We reserve the right to change our products and their specification as well as related user manuals, accompanying texts and documentation at all times and without advance notice, without obligation to report the change. Changes will be included in future manuals and do not constitute any obligations. There is no entitlement to revisions of delivered documents. The manual delivered with the product applies.

Hilscher Gesellschaft für Systemautomation mbH is not liable under any circumstances for direct, indirect, incidental or follow-on damage or loss of earnings resulting from the use of the information contained in this publication.

### 1.8.3 Exclusion of Liability

The software was produced and tested with utmost care by Hilscher Gesellschaft für Systemautomation mbH and is made available as is. No warranty can be assumed for the performance and flawlessness of the software for all usage conditions and cases and for the results produced when utilized by the user. Liability for any damages that may result from the use of the hardware or software or related documents, is limited to cases of intent or grossly negligent violation of significant contractual obligations. Indemnity claims for the violation of significant contractual obligations are limited to damages that are foreseeable and typical for this type of contract.

It is strictly prohibited to use the software in the following areas:

- for military purposes or in weapon systems;
- for the design, construction, maintenance or operation of nuclear facilities;
- in air traffic control systems, air traffic or air traffic communication systems;
- in life support systems;
- in systems in which failures in the software could lead to personal injury or injuries leading to death.

We inform you that the software was not developed for use in dangerous environments requiring fail-proof control mechanisms. Use of the software in such an environment occurs at your own risk. No liability is assumed for damages or losses due to unauthorized use.

### 1.8.4 Export

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different countries, in particular those of Germany and the USA. The software may not be exported to countries where this is prohibited by the United States Export Administration Act and its additional provisions. You are obligated to comply with the regulations at your personal responsibility. We wish to inform you that you may require permission from state authorities to export, re-export or import the product.

## 2 Fundamentals

### 2.1 General Access Mechanisms on netX Systems

This chapter explains the possible ways to access a Protocol Stack running on a netX system :

1. By accessing the Dual Port Memory Interface directly or via a driver.
2. By accessing the Dual Port Memory Interface via a shared memory.
3. By interfacing with the Stack Task of the Protocol Stack.

The picture below visualizes these three ways:

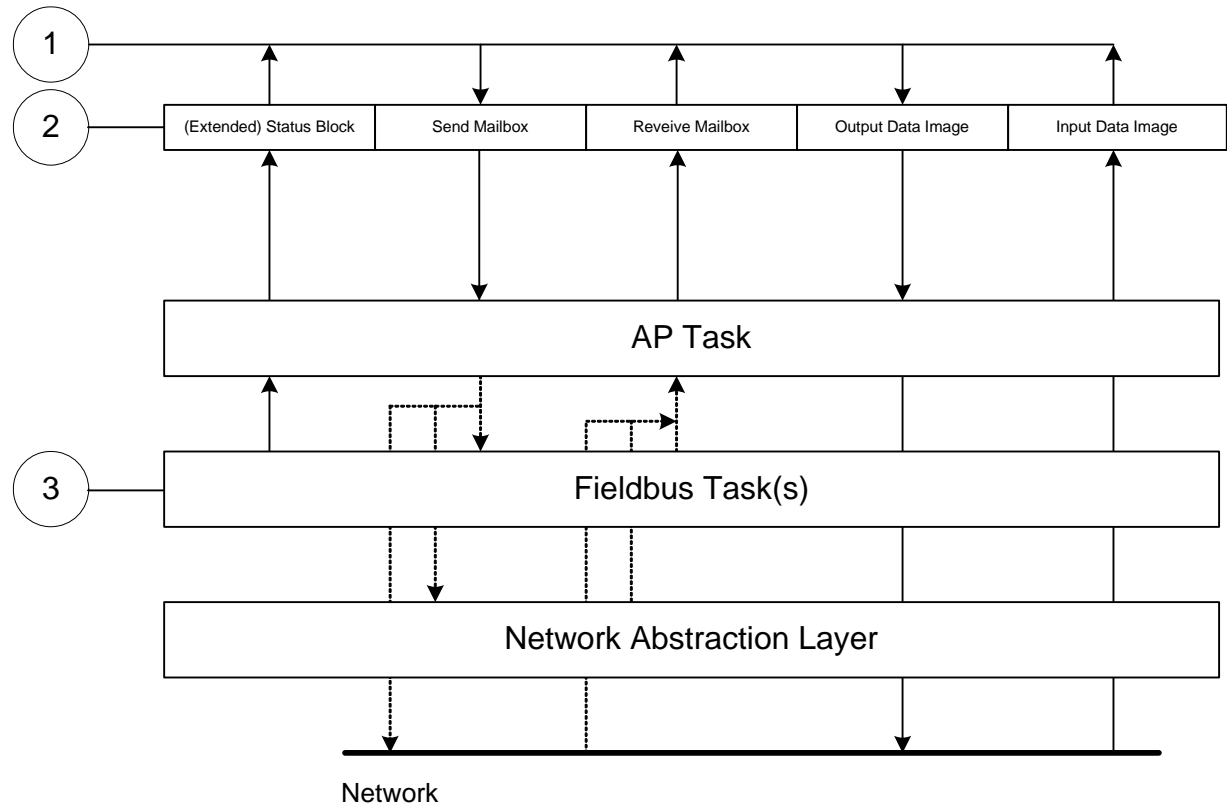


Figure 1: The 3 different Ways to access a Protocol Stack running on a netX System

This chapter explains how to program the stack (alternative 3) correctly while the next chapter describes accessing the protocol stack via the dual-port memory interface according to alternative 1 (and 2, if the user application is executed on the netX chip in the context of the rcX operating system and uses the virtual DPM).

Finally, chapter 6 titled “The Application Interface” describes the entire interface to the protocol stack in detail.

Depending on you choose the stack-oriented approach or the Dual Port Memory-based approach, you will need either the information given in this chapter or those of the next chapter to be able to work with the set of functions described in chapter 6. All of those functions use the four parameters `ulDest`, `ulSrc`, `ulDestId` and `ulSrcId`. This chapter and the next one inform about how to work with these important parameters.

## 2.2 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.

### 2.2.1 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

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_SENDBUFFER_FIFO/LIFO()` for sending a packet to the respective task.

### 2.2.2 Meaning of Source- and Destination-related Parameters

The meaning of the source- and destination-related parameters is explained in the following table:

Variable	Meaning
<code>ulDest</code>	Application mailbox used for confirmation
<code>ulSrc</code>	Queue handle returned by <code>TLR_QUE_IDENTIFY()</code> as described above.
<code>ulSrcId</code>	Used for addressing at a lower level

For more information about programming the AP task's stack queue, please refer to the Hilscher Task Layer Reference Model Manual. Especially the following sections might be of interest in this context:

1. Section 7 "Queue-Packets"
2. Section 10.1.9 "Queuing Mechanism"

## 2.3 Accessing the Protocol Stack via the Dual Port Memory Interface

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

### 2.3.1 Communication via Mailboxes

The mailbox of each communication channel has two areas that are used for non-cyclic message transfer to and from the netX.

- **Send Mailbox**  
Packet transfer from host system to netX firmware
- **Receive Mailbox**  
Packet transfer from netX firmware to host system

For more details about acyclic data transfer via mailboxes see section 3.2. The concept of using messages called packets in this context, is described in detail in section 3.2.1 “*General Structure of Messages or Packets for Non-Cyclic Data Exchange*” while the possible codes that may appear are listed in section 3.2.2. “*Status & Error Codes*”.

However, this section concentrates on correct addressing the mailboxes.

### 2.3.2 Using Source and Destination Variables correctly

#### 2.3.2.1 How to use ulDest for Addressing rcX and the netX Protocol Stack by the System and Channel Mailbox

The preferred way to address the netX operating system rcX is through the system mailbox; the preferred way to address a protocol stack is through its channel mailbox. All mailboxes, however, have a mechanism to route packets to a communication channel or the system channel, respectively. Therefore, the destination identifier *ulDest* in a packet header has to be filled in according to the targeted receiver. See the following example.

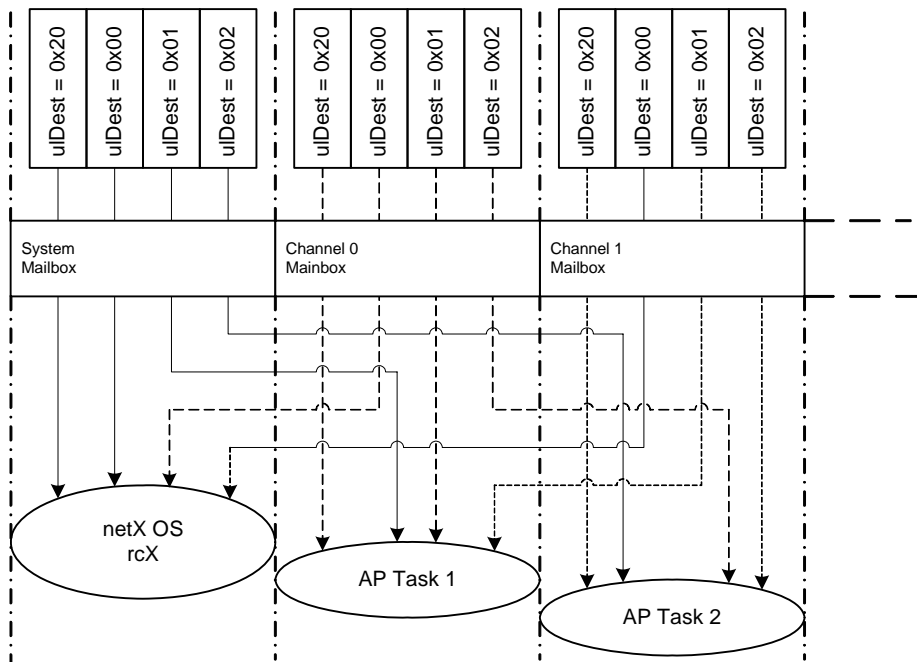


Figure 2: Use of *ulDest* in Channel and System Mailbox

For use in the destination queue handle, the tasks have been assigned to hexadecimal numerical values as described in the following table:

<b>ulDest</b>	<b>Description</b>
0x00000000	Packet is passed to the netX operating system rcX
0x00000001	Packet is passed to communication channel 0
0x00000002	Packet is passed to communication channel 1
0x00000003	Packet is passed to communication channel 2
0x00000004	Packet is passed to communication channel 3
0x00000020	Packet is passed to communication channel of the mailbox
else	Reserved, do not use

The picture and the table above both show the use of the destination identifier *ulDest*.

A remark on the special channel identifier 0x00000020 (= *Channel Token*): The Channel Token is valid for any mailbox. That way the application uses the same identifier for all packets without actually knowing which mailbox or communication channel is applied. The packet stays 'local'. The system mailbox is a little bit different, because it is used to communicate to the netX operating system rcX. The rcX has its own range of valid commands codes and differs from a communication channel.

Unless there is a reply packet, the netX operating system returns it to the same mailbox the request packet went through. Consequently, the host application has to return its reply packet to the mailbox the request was received from.

### 2.3.2.2 How to use `ulSrc` and `ulSrcId`

Generally, a netX protocol stack can be addressed through its communication channel mailbox. The example below shows how a host application addresses a protocol stack running in the context of a netX chip. The application is identified by a number (#444 in this example). The application consists of three processes identified by the numbers #11, #22 and #33. These processes communicate through the channel mailbox with the AP task of the protocol stack. Have a look at the following image:

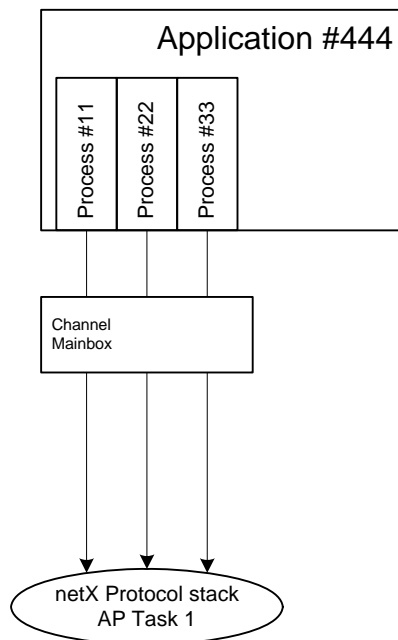


Figure 3: Using `ulSrc` and `ulSrcId`

**Example**

This example applies to command messages initiated by a process in the context of the host application. If the process #22 sends a packet through the channel mailbox to the AP task, the packet header has to be filled in as follows:

Object	Variable Name	Numeric Value	Explanation
Destination Queue Handle	<code>ulDest</code>	= 32 (0x00000020)	This value needs always to be set to 0x00000020 (the channel token) when accessing the protocol stack via the local communication channel mailbox.
Source Queue Handle	<code>ulSrc</code>	= 444	Denotes the host application (#444).
Destination Identifier	<code>ulDestId</code>	= 0	In this example it is not necessary to use the destination identifier.
Source Identifier	<code>ulSrcId</code>	= 22	Denotes the process number of the process within the host application and needs therefore to be supplied by the programmer of the host application.

For packets through the channel mailbox, the application uses 32 (= 0x20, *Channel Token*) for the destination queue handler `ulDest`. The source queue handler `ulSrc` and the source identifier `ulSrcId` are used to identify the originator of a packet. The destination identifier `ulDestId` can be used to address certain resources in the protocol stack. It is not used in this example. The source queue handler `ulSrc` has to be filled in. Therefore its use is mandatory; the use of `ulSrcId` is optional.

The netX operating system passes the request packet to the protocol stack's AP task. The protocol stack then builds a reply to the packet and returns it to the mailbox. The application has to make sure that the packet finds its way back to the originator (process #22 in the example).

**2.3.2.3 How to Route rcX Packets**

To route an rcX packet the source identifier `ulSrcId` and the source queues handler `ulSrc` in the packet header hold the identification of the originating process. The router saves the original handle from `ulSrcId` and `ulSrc`. The router uses a handle of its own choices for `ulSrcId` and `ulSrc` before it sends the packet to the receiving process. That way the router can identify the corresponding reply packet and matches the handle from that packet with the one stored earlier. Now the router replaces its handles with the original handles and returns the packet to the originating process.

### 2.3.3 Obtaining useful Information about the Communication Channel

A communication channel represents a part of the Dual Port Memory and usually consists of the following elements:

- **Output Data Image**  
is used to transfer cyclic process data to the network (normal or high-priority)
- **Input Data Image**  
is used to transfer cyclic process data from the network (normal or high-priority)
- **Send Mailbox**  
is used to transfer non-cyclic data to the netX
- **Receive Mailbox**  
is used to transfer non-cyclic data from the netX
- **Control Block**  
allows the host system to control certain channel functions
- **Common Status Block**  
holds information common to all protocol stacks
- **Extended Status Block**  
holds protocol specific network status information

This section describes a procedure how to obtain useful information for accessing the communication channel(s) of your netX device and to check if it is ready for correct operation.

Proceed as follows:

- 1) Start with reading the channel information block within the system channel (usually starting at address 0x0030).
- 2) Then you should check the hardware assembly options of your netX device. They are located within the system information block following offset 0x0010 and stored as data type `UINT16`. The following table explains the relationship between the offsets and the corresponding xC Ports of the netX device:

0x0010	Hardware Assembly Options for xC Port[0]
0x0012	Hardware Assembly Options for xC Port[1]
0x0014	Hardware Assembly Options for xC Port[2]
0x0016	Hardware Assembly Options for xC Port[3]

Check each of the hardware assembly options whether its value has been set to `RCX_HW_ASSEMBLY_PROFIBUS = 0x0050`. If true, this denotes that this xCPort is suitable for running the Profibus protocol stack. Otherwise, this port is designed for another communication protocol. In most cases, xC Port[2] will be used for field-bus systems, while xC Port[0] and xC Port[1] are normally used for Ethernet communication.

- 3) You can find information about the corresponding communication channel (0...3) under the following addresses:

0x0050	Communication Channel 0
0x0060	Communication Channel 1
0x0070	Communication Channel 2
0x0080	Communication Channel 3

In devices which support only one communication system which is usually the case (either a single field-bus system or a single standard for Industrial-Ethernet communication), always communication channel 0 will be used. In devices supporting more than one communication system you should also check the other communication channels.

- 4) There you can find such information as the ID (containing channel number and port number) of the communication channel, the size and the location of the handshake cells, the overall number of blocks within the communication channel and the size of the channel in bytes. Evaluate this information precisely in order to access the communication channel correctly.

The information is delivered as follows:

Size of Channel in Bytes

Address	Data Type	Description
0x0050	UINT8	Channel Type = COMMUNICATION (must have the fixed value <code>define RCX_CHANNEL_TYPE_COMMUNICATION = 0x05</code> )
0x0051	UINT8	ID (Channel Number, Port Number)
0x0052	UINT8	Size / Position Of Handshake Cells
0x0053	UINT8	Total Number Of Blocks Of This Channel
0x0054	UINT32	Size Of Channel In Bytes
0x0058	UINT8[8]	Reserved (set to zero)

These addresses correspond to communication channel 0, for communication channels 1, 2 and 3 you have to add an offset of 0x0010, 0x0020 or 0x0030 to the address values, respectively.

- 5) Finally, you can access the communication channel using the addresses you determined previously. For more information how to do this, please refer to the netX DPM Manual, especially section 3.2 "Communication Channel".

## 2.4 Client/Server Mechanism

### 2.4.1 Application as Client

The host application may send request packets to the netX firmware at any time (transition 1 ⇒ 2). Depending on the protocol stack running on the netX, parallel packets are not permitted (see protocol specific manual for details). The netX firmware sends a confirmation packet in return, signaling success or failure (transition 3 ⇒ 4) while processing the request.

The host application has to register with the netX firmware in order to receive indication packets (transition 5 ⇒ 6). Depending on the protocol stack, this is done either implicit (if application opens a TCP/UDP socket) or explicit (if application wants to receive unsolicited DPV1 packets). Details on when and how to register for certain events is described in the protocol specific manual. Depending on the command code of the indication packet, a response packet to the netX firmware may or may not be required (transition 7 ⇒ 8).

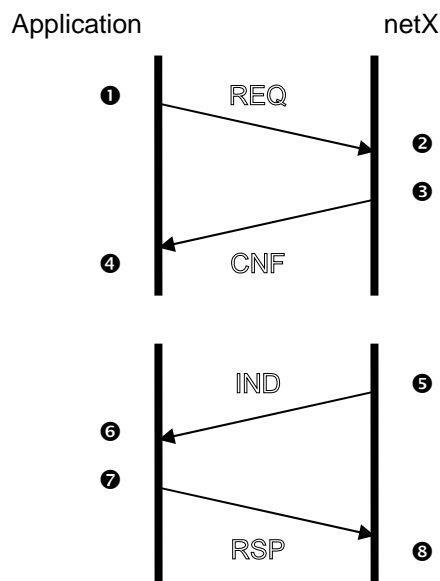


Figure 4: Transition Chart Application as Client

- ① ② The host application sends request packets to the netX firmware.
- ③ ④ The netX firmware sends a confirmation packet in return.
- ⑤ ⑥ The host application receives indication packets from the netX firmware.
- ⑦ ⑧ The host application sends response packet to the netX firmware (may not be required).

REQ Request                      CNF Confirmation

IND Indication                    RSP Response

## 2.4.2 Application as Server

The host application has to register with the netX firmware in order to receive indication packets. Depending on the protocol stack, this is done either implicit (if application opens a TCP/UDP socket) or explicit (if application wants to receive unsolicited DPV1 packets). Details on when and how to register for certain events is described in the protocol specific manual.

When an appropriate event occurs and the host application is registered to receive such a notification, the netX firmware passes an indication packet through the mailbox (transition 1 ⇒ 2). The host application is expected to send a response packet back to the netX firmware (transition 3 ⇒ 4).

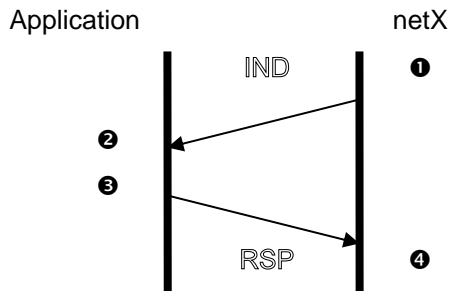


Figure 5: Transition Chart Application as Server

① ② The netX firmware passes an indication packet through the mailbox.

③ ④ The host application sends response packet to the netX firmware.

IND Indication                      RSP Response

## 3 Dual-Port-Memory

All data in the dual-port memory is structured in blocks. According to their functions, these blocks use different data transfer mechanisms. For example, data transfer through mailboxes uses a synchronized handshake mechanism between host system and netX firmware. The same is true for IO data images, when a buffered handshake mode is configured. Other blocks, like the status block, are read by the host application and use no synchronization mechanism.

Types of blocks in the dual-port memory are outlined below:

- **Mailbox**  
transfer non-cyclic messages or packages with a header for routing information
- **Data Area**  
holds the process image for cyclic IO data or user defined data structures
- **Control Block**  
is used to signal application related state to the netX firmware
- **Status Block**  
holds information regarding the current network state
- **Change of State**  
collection of flags, that initiate execution of certain commands or signal a change of state

### 3.1 Cyclic Data (Input/Output Data)

The input block holds the process data image received **from** the network whereas the output block holds data sent **to** the network.

Process data transfer through the data blocks can be synchronized by using a handshake mechanism (configurable). If in uncontrolled mode, the protocol stack updates the process data in the input and output data image in the dual-port memory for each valid bus cycle. No handshake bits are evaluated and no buffers are used. The application can read or write process data at any given time without obeying the synchronization mechanism otherwise carried out via handshake location. This transfer mechanism is the simplest method of transferring process data between the protocol stack and the application. This mode can only guarantee data consistency over a byte.

For the controlled / buffered mode, the protocol stack updates the process data in the internal input buffer for each valid bus cycle. Each IO block uses handshake bits for access synchronization. Input and output data block handshake operates independently from each other. When the application toggles the input handshake bit, the protocol stack copies the data from the internal buffer into the input data image of the dual-port memory. Now the application can copy data from the dual-port memory and then give control back to the protocol stack by toggling the appropriate input handshake bit. When the application/driver toggles the output handshake bit, the protocol stack copies the data from the output data image of the dual-port memory into the internal buffer. From there the data is transferred to the network. The protocol stack toggles the handshake bits back, indicating to the application that the transfer is finished and a new data exchange cycle may start. This mode guarantees data consistency over both input and output area.

### 3.1.1 Input Data Image

The input data block is used by fieldbus and industrial Ethernet protocols that utilize a cyclic data exchange mechanism. The input data image is used to transfer cyclic data **from** the network.

The default size of the input data image is 5760 byte. An output and input data block may or may not be available in the dual-port memory. They are always available in the default memory map (see the netX Dual-Port Memory Manual).



**Note:** 48 byte are used for status information (16 byte for list of configured slaves, 16 byte for list of activated slaves and 16 byte for list of slaves with faults or errors). Therefore the maximum amount of really usable input data is 5712 byte.

The contents of these 48 byte is identical to the contents of the second part of the Extended Status Block beginning at address 0x0100, see *Table 23: Extended Status Block for PROFIBUS DP-Master – Second part (State Field Definition Block)*.

Input Data Image			
Offset	Type	Name	Description
0x2680	UINT8	abPd0Input [ 5760 ]	Input Data Image Cyclic Data From The Network

Table 4: Input Data Image



**Note:** In case of a network fault (e.g. a disconnected network cable), a slave firmware keeps the last state of the input data image. As soon as the firmware detects the network fault, it clears the *Communicating* flag in netX communication flags (see section 3.2.2.1); the input data should not be evaluated.

### 3.1.2 Process Data Output

The output data block is used by fieldbus and industrial Ethernet protocols that utilize a cyclic data exchange mechanism. The output data Image is used to transfer cyclic data **to** the network.

The default size of the output data image is 5760 byte (including 48 bytes of status information). An output data block may or may not be available in the dual-port memory. They are always available in the default memory map (see netX DPM Manual).

Output Data Image			
Offset	Type	Name	Description
0x1000	UINT8	abPd0Output [ 5760 ]	Output Data Image Cyclic Data To The Network

Table 5: Output Data Image

## 3.2 Acyclic Data (Mailboxes)

The mailbox of each communication channel has two areas that are used for non-cyclic message transfer.

- **Send Mailbox**  
Packet transfer from host system to firmware
- **Receive Mailbox**  
Packet transfer from firmware to host system

The send and receive mailbox areas are used by field bus protocols providing a non-cyclic data exchange mechanism. Another use of the mailbox system is to allow access to the firmware running on the netX chip itself for diagnostic and identification purposes. The send mailbox is used to transfer cyclic data **to** the network or **to** the firmware. The receive mailbox is used to transfer cyclic data **from** the network or **from** the firmware.

A send/receive mailbox may or may not be available in the communication channel. It depends on the function of the firmware whether or not a mailbox is needed. The location of the system mailbox and the channel mailbox is described in the netX DPM Interface Manual.



**Note:** Each mailbox can hold one packet at a time. The netX firmware stores packets that are not retrieved by the host application in a packet queue. This queue has limited space and may fill up so new packets maybe lost. To avoid these data loss situations, it is strongly recommended to empty the mailbox frequently, even if packets are not expected by the host application. Unexpected command packets should be returned to the sender with an Unknown Command in the status field; unexpected reply messages can be discarded.

### 3.2.1 General Structure of Messages or Packets for Non-Cyclic Data Exchange

The non-cyclic packets through the netX mailbox have the following structure:

Structure Information				
Area	Variable	Type	Value / Range	Description
tHead	Structure Information			
	ulDest	UINT32		Destination Queue Handle
	ulSrc	UINT32		Source Queue Handle
	ulDestId	UINT32		Destination Queue Reference
	ulSrcId	UINT32		Source Queue Reference
	ulLen	UINT32		Packet Data Length (In Bytes)
	ulId	UINT32		Packet Identification As Unique Number
	ulSta	UINT32		Status / Error Code
	ulCmd	UINT32		Command / Response
	ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information	
tData	Structure Information			
	...	...		User Data Specific To The Command

Some of the fields are mandatory; some are conditional; others are optional. However, the size of a packet is always at least 10 double-words or 40 bytes. Depending on the command, a packet may or may not have a data field. If present, the content of the data field is specific to the command, respectively the reply.

#### Destination Queue Handler

The *ulDest* field identifies a task queue in the context of the netX firmware. The task queue represents the final receiver of the packet and is assigned to a protocol stack. The *ulDest* field has to be filled out in any case. Otherwise, the netX operating system cannot route the packet. This field is mandatory.

#### Source Queue Handler

The *ulSrc* field identifies the sender of the packet. In the context of the netX firmware (inter-task communication) this field holds the identifier of the sending task. Usually, a driver uses this field for its own handle, but it can hold any handle of the sending process. Using this field is mandatory. The receiving task does not evaluate this field and passes it back unchanged to the originator of the packet.

### Destination Identifier

The *ulDestId* field identifies the destination of an unsolicited packet from the netX firmware to the host system. It can hold any handle that helps to identify the receiver. Therefore, its use is mandatory for unsolicited packets. The receiver of unsolicited packets has to register for this.

### Source Identifier

The *ulSrcId* field identifies the originator of a packet. This field is used by a host application, which passes a packet from an external process to an internal netX task. The *ulSrcId* field holds the handle of the external process. When netX operating system returns the packet, the application can identify the packet and returns it to the originating process. The receiving task on the netX does not evaluate this field and passes it back unchanged. For inter-task communication, this field is not used.

### Length of Data Field

The *ulLen* field holds the size of the data field in bytes. It defines the total size of the packet's payload that follows the packet's header. The size of the header is not included in *ulLen*. So the total size of a packet is the size from *ulLen* plus the size of packet's header. Depending on the command, a data field may or may not be present in a packet. If no data field is included, the length field is set to zero.

### Identifier

The *ulId* field is used to identify a specific packet among others of the same kind. That way the application or driver can match a specific reply or confirmation packet to a previous request packet. The receiving task does not change this field and passes it back to the originator of the packet. Its use is optional in most of the cases, but it is mandatory for sequenced packets.

#### Example:

Downloading big amounts of data that does not fit into a single packet. For a sequence of packets the identifier field is incremented by one for every new packet.

### Status / Error Code

The *ulState* field is used in response or confirmation packets. It informs the originator of the packet about success or failure of the execution of the command. The field may be also used to hold status information in a request packet.

### Command / Response

The *ulCmd* field holds the command code or the response code, respectively. The command/response is specific to the receiving task. If a task is not able to execute certain commands, it will return the packet with an error indication. A command is always even (the least significant bit is zero). In the response packet, the command code is incremented by one indicating a confirmation to the request packet.

### Extension

The extension field *ulExt* is used for controlling packets that are sent in a sequenced manner. The extension field indicates the first, last or a packet of a sequence. If sequencing is not required, the extension field is not used and set to zero.

### Routing Information

The *ulRout* field is used internally by the netX firmware only. It has no meaning to a driver type application and therefore set to zero.

### User Data Field

This field contains data related to the command specified in *ulCmd* field. Depending on the command, a packet may or may not have a data field. The length of the data field is given in the *ulLen* field.

## 3.2.2 Status & Error Codes

The following status and error codes can be returned in *ulState*: List of codes see manual named *netX Dual-Port Memory Interface*.

## 3.2.3 Differences between System and Channel Mailboxes

The mailbox system on netX provides a non-cyclic data transfer channel for fieldbus and industrial Ethernet protocols. Another use of the mailbox is allowing access to the firmware running on the netX chip itself for diagnostic purposes. There is always a send and a receive mailbox. Send and receive mailboxes utilize handshake bits to synchronize these data or diagnostic packages through the mailbox. There is a pair of handshake bits for both the send and receive mailbox.

The netX operating system rcX only uses the system mailbox.

- The *system mailbox*, however, has a mechanism to route packets to a communication channel.
- A *channel mailbox* passes packets to its own protocol stack only.

## 3.2.4 Send Mailbox

The send mailbox area is used by protocols utilizing a non-cyclic data exchange mechanism. Another use of the mailbox system is to provide access to the firmware running on the netX chip itself. The **send** mailbox is used to transfer non-cyclic data **to** the network or **to** the protocol stack.

The size is 1596 bytes for the send mailbox in the default memory layout. The mailbox is accompanied by counters that hold the number of packages that can be accepted.

### 3.2.5 Receive Mailbox

The receive mailbox area is used by protocols utilizing a non-cyclic data exchange mechanism. Another use of the mailbox system is to provide access to the firmware running on the netX chip itself. The **receive** mailbox is used to transfer non-cyclic data **from** the network or **from** the protocol stack.

The size is 1596 bytes for the receive mailbox in the default memory layout. The mailbox is accompanied by counters that hold the number of waiting packages (for the receive mailbox).

### 3.2.6 Channel Mailboxes (Details of Send and Receive Mailboxes)

Master Status			
Offset	Type	Name	Description
0x0200	UINT16	usPackagesAccepted	Packages Accepted Number of Packages that can be Accepted
0x0202	UINT16	usReserved	Reserved Set to 0
0x0204	UINT8	abSendMbx[ 1596 ]	Send Mailbox Non Cyclic Data To The Network or to the Protocol Stack
0x0840	UINT16	usWaitingPackages	Packages waiting Counter of packages that are waiting to be processed
0x0842	UINT16	usReserved	Reserved Set to 0
0x0844	UINT8	abRecvMbx[ 1596 ]	Receive Mailbox Non Cyclic Data <b>from</b> the network or <b>from</b> the protocol stack

Table 6: Channel Mailboxes

#### Channel Mailboxes Structure

```
typedef struct tagNETX_SEND_MAILBOX_BLOCK
{
  UINT16 usPackagesAccepted;
  UINT16 usReserved;
  UINT8 abSendMbx[ 1596 ];
} NETX_SEND_MAILBOX_BLOCK;
typedef struct tagNETX_RECV_MAILBOX_BLOCK
{
  UINT16 usWaitingPackages;
  UINT16 usReserved;
  UINT8 abRecvMbx[ 1596 ];
} NETX_RECV_MAILBOX_BLOCK;
```

### 3.3 Status

A status block is present within the communication channel. It contains information about network and task related issues. In some respects, status and control block are used together in order to exchange information between host application and netX firmware. The application reads a status block whereas the control block is written by the application. Both status and control block have registers that use the *Change of State* mechanism (see also section 2.2.1 of the *netX Dual-Port-Memory manual*).

#### 3.3.1 Common Status

The Common Status Block contains information that is the same for all communication channels. The start offset of this block depends on the size and location of the preceding blocks. The status block is always present in the dual-port memory.

##### 3.3.1.1 All Implementations

The structure outlined below is common to all protocol stacks.

##### Common Status Structure Definition

Common Status			
Offset	Type	Name	Description
0x0010	UINT32	ulCommunicationCOS	<u>Communication Change of State</u> READY, RUN, RESET REQUIRED, NEW, CONFIG AVAILABLE, CONFIG LOCKED
0x0014	UINT32	ulCommunicationState	<u>Communication State</u> NOT CONFIGURED, STOP, IDLE, OPERATE
0x0018	UINT32	ulCommunicationError	<u>Communication Error</u> Unique Error Number According to Protocol Stack
0x001C	UINT16	usVersion	<u>Version</u> Version Number of this Diagnosis Structure
0x001E	UINT16	usWatchdogTime	<u>Watchdog Timeout</u> Configured Watchdog Time
0x0020	UINT16	usHandshakeMode	Handshake Mode Process Data Transfer Mode (see netX DPM Interface Manual)
0x0022	UINT16	usReserved	Reserved Set to 0
0x0024	UINT32	ulHostWatchdog	<u>Host Watchdog</u> Joint Supervision

			Mechanism Protocol Stack Writes, Host System Reads
<b>0x0028</b>	UINT32	ulErrorCount	<u>Error Count</u> Total Number of Detected Error Since Power-Up or Reset
<b>0x002C</b>	UINT32	ulErrorLogInd	<u>Error Log Indicator</u> Total Number Of Entries In The Error Log Structure (not supported yet)
<b>0x0030</b>	UINT32	ulReserved[2]	<u>Reserved</u> Set to 0

Table 7: Common Status Structure Definition

### Common Status Block Structure Reference

```
typedef struct NETX_COMMON_STATUS_BLOCK_Ttag
{
    UINT32    ulCommunicationCOS;
    UINT32    ulCommunicationState;
    UINT32    ulCommunicationError;
    UINT16    usVersion;
    UINT16    usWatchdogTime;
    UINT16    ausReserved[2];
    UINT32    ulHostWatchdog;
    UINT32    ulErrorCount;
    UINT32    ulErrorLogInd;
    UINT32    ulReserved[2];
    union
    {
        {
            NETX_MASTER_STATUS_T    tMasterStatus;    /* for master implementation */
            UINT32                    aulReserved[6];    /* otherwise reserved */
        } unStackDepended;
    }
} NETX_COMMON_STATUS_BLOCK_T;
```

## Common Status Block Structure Reference

```
typedef struct NETX_COMMON_STATUS_BLOCK_Ttag
{
    UUINT32    ulCommunicationCOS;
    UUINT32    ulCommunicationState;
    UUINT32    ulCommunicationError;
    UUINT16    usVersion;
    UUINT16    usWatchdogTime;
    UUINT16    ausReserved[2];
    UUINT32    ulHostWatchdog;
    UUINT32    ulErrorCount;
    UUINT32    ulErrorLogInd;
    UUINT32    ulReserved[2];
    union
    {
        {
            NETX_MASTER_STATUS_T    tMasterStatus;    /* for master implementation */
            UUINT32    aulReserved[6];    /* otherwise reserved */
        }
    } unStackDepended;
} NETX_COMMON_STATUS_BLOCK_T;
```

### Communication Change of State (All Implementations)

The communication change of state register contains information about the current operating status of the communication channel and its firmware. Every time the status changes, the netX protocol stack toggles the *netX Change of State Command* flag in the netX communication flags register (see section 3.2.2.1 of the netX DPM Interface Manual). The application then has to toggle the *netX Change of State Acknowledge* flag back acknowledging the new state (see section 3.2.2.2 of the netX DPM Interface Manual).

ulCommunicationCOS - netX writes, Host reads		
Bit	Short name	Name
D31..D7	unused, set to zero	
D6	Restart Required Enable	RCX_COMM_COS_RESTART_REQUIRED_ENABLE
D5	Restart Required	RCX_COMM_COS_RESTART_REQUIRED
D4	Configuration New	RCX_COMM_COS_CONFIG_NEW
D3	Configuration Locked	RCX_COMM_COS_CONFIG_LOCKED
D2	Bus On	RCX_COMM_COS_BUS_ON
D1	Running	RCX_COMM_COS_RUN
D0	Ready	RCX_COMM_COS_READY

Table 8: Communication State of Change

**Communication Change of State Flags (netX System ⇌ Application)**

Bit	Definition / Description
0	Ready (RCX_COMM_COS_READY) 0 - ... 1 - The <i>Ready</i> flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the <i>Running</i> flag is set, too.
1	Running (RCX_COMM_COS_RUN) 0 - ... 1 -The <i>Running</i> flag is set when the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the <i>Ready</i> flag and the <i>Running</i> flag are set.
2	Bus On (RCX_COMM_COS_BUS_ON) 0 - ... 1 -The <i>Bus On</i> flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network; if cleared, the permission was denied and the protocol stack will not open network connections.
3	Configuration Locked (RCX_COMM_COS_CONFIG_LOCKED) 0 - ... 1 -The <i>Configuration Locked</i> flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the <i>Lock Configuration</i> flag in the control block (see page 43).
4	Configuration New (RCX_COMM_COS_CONFIG_NEW) 0 - ... 1 -The <i>Configuration New</i> flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the <i>Restart Required</i> flag.
5	Restart Required (RCX_COMM_COS_RESTART_REQUIRED) 0 - ... 1 -The <i>Restart Required</i> flag is set when the channel firmware requests to be restarted. This flag is used together with the <i>Restart Required Enable</i> flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
6	Restart Required Enable (RCX_COMM_COS_RESTART_REQUIRED_ENABLE) 0 - ... 1 - The <i>Restart Required Enable</i> flag is used together with the <i>Restart Required</i> flag above. If set, this flag enables the execution of the <i>Restart Required</i> command in the netX firmware (for details on the <i>Enable</i> mechanism see section 2.3.2 of the netX DPM Interface Manual)).
7 ... 31	Reserved, set to 0

Table 9: Meaning of Communication Change of State Flags

**Communication State (All Implementations)**

The communication state field contains information regarding the current network status of the communication channel. Depending on the implementation, all or a subset of the definitions below is supported.

■ UNKNOWN	#define RCX_COMM_STATE_UNKNOWN	0x00000000
■ NOT_CONFIGURED	#define RCX_COMM_STATE_NOT_CONFIGURED	0x00000001
■ STOP	#define RCX_COMM_STATE_STOP	0x00000002
■ IDLE	#define RCX_COMM_STATE_IDLE	0x00000003
■ OPERATE	#define RCX_COMM_STATE_OPERATE	0x00000004

**Communication Channel Error (All Implementations)**

This field holds the current error code of the communication channel. If the cause of error is resolved, the communication error field is set to zero (= RCX\_SYS\_SUCCESS) again. Not all of the error codes are supported in every implementation. Protocol stacks may use a subset of the error codes below.

■ SUCCESS	#define RCX_SYS_SUCCESS	0x00000000
-----------	-------------------------	------------

**Runtime Failures**

■ WATCHDOG TIMEOUT	#define RCX_E_WATCHDOG_TIMEOUT	0xC000000C
--------------------	--------------------------------	------------

**Initialization Failures**

■ (General) INITIALIZATION FAULT	#define RCX_E_INIT_FAULT	0xC0000100
■ DATABASE ACCESS FAILED	#define RCX_E_DATABASE_ACCESS_FAILED	0xC0000101

**Configuration Failures**

■ NOT CONFIGURED	#define RCX_E_NOT_CONFIGURED	0xC0000119
■ (General) CONFIGURATION FAULT	#define RCX_E_CONFIGURATION_FAULT	0xC0000120
■ INCONSISTENT DATA SET	#define RCX_E_INCONSISTENT_DATA_SET	0xC0000121
■ DATA SET MISMATCH	#define RCX_E_DATA_SET_MISMATCH	0xC0000122
■ INSUFFICIENT LICENSE	#define RCX_E_INSUFFICIENT_LICENSE	0xC0000123
■ PARAMETER ERROR	#define RCX_E_PARAMETER_ERROR	0xC0000124
■ INVALID NETWORK ADDRESS	#define RCX_E_INVALID_NETWORK_ADDRESS	0xC0000125
■ NO SECURITY MEMORY	#define RCX_E_NO_SECURITY_MEMORY	0xC0000126

**Network Failures**

■ (General) NETWORK FAULT	#define RCX_COMM_NETWORK_FAULT	0xC0000140
■ CONNECTION CLOSED	#define RCX_COMM_CONNECTION_CLOSED	0xC0000141
■ CONNECTION TIMED OUT	#define RCX_COMM_CONNECTION_TIMEOUT	0xC0000142
■ LONELY NETWORK	#define RCX_COMM_LONELY_NETWORK	0xC0000143
■ DUPLICATE NODE	#define RCX_COMM_DUPLICATE_NODE	0xC0000144
■ CABLE DISCONNECT	#define RCX_COMM_CABLE_DISCONNECT	0xC0000145

**Version (All Implementations)**

The version field holds version of this structure. It starts with one; zero is not defined.

■ STRUCTURE VERSION	#define RCX_STATUS_BLOCK_VERSION	0x0001
---------------------	----------------------------------	--------

**Watchdog Timeout (All Implementations)**

This field holds the configured watchdog timeout value in milliseconds. The application may set its watchdog trigger interval accordingly. If the application fails to copy the value from the host watchdog location to the device watchdog location, the protocol stack will interrupt all network connections immediately regardless of their current state. For details, see section 4.13 of the netX DPM Interface Manual.

**Host Watchdog (All Implementations)**

The protocol stack supervises the host system using the watchdog function. If the application fails to copy the value from the device watchdog location (section 3.2.5 of the netX DPM Interface Manual) to the host watchdog location (section 3.2.4 of the netX DPM Interface Manual), the protocol stack assumes that the host system has some sort of problem and shuts down all network connections. For details on the watchdog function, refer to section 4.13 of the netX DPM Interface Manual.

**Error Count (All Implementations)**

This field holds the total number of errors detected since power-up, respectively after reset. The protocol stack counts all sorts of errors in this field no matter if they were network related or caused internally.

**Error Log Indicator (All Implementations)**

Not supported yet: The error log indicator field holds the number of entries in the internal error log. If all entries are read from the log, the field is set to zero.

### 3.3.1.2 Master Implementation

In addition to the common status block as outlined in the previous section, a master firmware maintains the following structure.

#### Master Status Structure Definition

Master Status			
Offset	Type	Name	Description
0x0010	Structure	See common structure in table <i>Common Status Block</i>	
0x0038	UINT32	ulSlaveState	<u>Slave State</u> OK, FAILED (At Least One Slave)
0x003C	UINT32	ulSlaveErrLogInd	<u>Slave Error Log Indicator</u> Slave Diagnosis Data Available: EMPTY, AVAILABLE
0x0040	UINT32	ulNumOfConfigSlaves	<u>Configured Slaves</u> Number of Configured Slaves On The Network
0x0044	UINT32	ulNumOfActiveSlaves	<u>Active Slaves</u> Number of Slaves Running Without Problems
0x0048	UINT32	ulNumOfDiagSlaves	<u>Faulted Slaves</u> Number of Slaves Reporting Diagnostic Issues
0x004C	UINT32	ulReserved	<u>Reserved</u> Set to 0

Table 10: Master Status Structure Definition

```
typedef struct NETX_MASTER_STATUS_Ttag
{
    UINT32 ulSlaveState;
    UINT32 ulSlaveErrLogInd;
    UINT32 ulNumOfConfigSlaves;
    UINT32 ulNumOfActiveSlaves;
    UINT32 ulNumOfDiagSlaves;
    UINT32 ulReserved;
} NETX_MASTER_STATUS_T;
```

### Slave State

The slave state field is available for master implementations only. It indicates whether the master is in cyclic data exchange to all configured slaves. In case there is at least one slave missing or if the slave has a diagnostic request pending, the status is set to *FAILED*. For protocols that support non-cyclic communication only, the slave state is set to *OK* as soon as a valid configuration is found.

Status and Error Codes		
Code (Symbolic Constant)	Numerical Value	Meaning
RCX_SLAVE_STATE_UNDEFINED	0x00000000	UNDEFINED
RCX_SLAVE_STATE_OK	0x00000001	OK
RCX_SLAVE_STATE_FAILED	0x00000002	FAILED (at least one slave)
Others are reserved		

Table 11: rcX Status and Error Codes/Slave

### Slave Error Log Indicator

The error log indicator field holds the number of entries in the internal error log. If all entries are read from the log, the field is set to zero. This function is not supported yet.

### Number of Configured Slaves

The firmware maintains a list of slaves to which the master has to open a connection. This list is derived from the configuration database created by SYCON.net (see section 6.1 of the netX Dual-Port Memory Manual). This field holds the number of configured slaves.

### Number of Active Slaves

The firmware maintains a list of slaves to which the master has successfully opened a connection. Ideally, the number of active slaves is equal to the number of configured slaves. For certain fieldbus systems it could be possible that the slave is shown as activated, but still has a problem in terms of a diagnostic issue. This field holds the number of active slaves.

### Number of Faulted Slaves

If a slave encounters a problem, it can provide an indication of the new situation to the master in certain fieldbus systems. As long as those indications are pending and not serviced, the field holds a value unequal zero. If no more diagnostic information is pending, the field is set to zero.

#### 3.3.1.3 Slave Implementation

The slave firmware only uses the common structure as outlined in section 3.3.1.1 of this document.

### 3.3.2 Extended Status

The content of the channel specific extended status block is specific to the implementation. Depending on the protocol, a status area may or may not be present in the dual-port memory. It is always available in the default memory map (see section 3.2.1 of netX Dual-Port Memory Manual).

#### netX Extended Status Field Definition 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;
```

Extended Status Block			
Offset	Type	Name	Description
0x0050	UINT8[ ]  (the first 64 bytes correspond to PROFIBUS_APM_EXT_STATUS_T)	abExtendedStatus[172]	Area containing PROFIBUS-DP-related information. See <i>Table 13: Extended Status Block for PROFIBUS DP-Master – First part</i> below  (first 64 bytes of abExtendedStatus)
0x0090	UINT8[ ]		Reserved area, currently unused, (rest of abExtendedStatus)
0x00FC	Structure  NETX_EXTENDED_STATE_FIELD_T	tExtStateField	Structure to define Status Fields and their Properties. Status type and properties are specific to protocol implementation

Table 12: Extended Status Block



**Note:** Each offset is always related to the begin of correspondent channel start

The definition of the first structure remains specific to correspondent protocol and contains additional information about network status (i.e. flags, error counters, events etc.).

The second structure begins at offset 0x00FC and 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.

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

At address **0x0050**, the following data structure is stored:

```
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 Block for PROFIBUS DP-Master – First part			
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)
0x53	unsigned char	bErr_Event	Error event
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 area
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 13: Extended Status Block for PROFIBUS DP-Master – First part

The single items of the Extended Status Block for Profibus DP Master have the following meaning

■ bGlobalBits/ Global Bits

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

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bits 0
Unused		MSK_PRO FIBUS_A PM_EXT_ STA_TOU T	MSK_PROF IBUS_APM _EXT_STA _NRDY	MSK_PRO FIBUS_A PM_EXT_ STA_FAT L_ERR	MSK_PRO FIBUS_A PM_EXT_ STA_NEX C_ERR	MSK_PROF IBUS_APM _EXT_STA _ACLR_ER R	MSK_PRO FIBUS_A PM_EXT_ STA_CTR L_ERR

Table 14: Contents of Global Bits Variable

This bit field serves as collective display of global notifications. Notified errors can either occur at the device itself or at the slaves. To distinguish the different errors the variable bErr\_Rem\_Adr contains the error location (i.e. bus address), while the variable bErr\_Event specifies the corresponding error number. If more than one error is determined, the error location will always show the lowest faulty bus address.

In detail these bits have the following meaning:

Error type	Explanation
MSK_PROFIBUS_APM_EXT_STA_CTRL_ERR	<b>CONTROL-ERROR:</b> This error is caused by incorrect parameterization.
MSK_PROFIBUS_APM_EXT_STA_ACLR_ERR	<b>AUTO-CLEAR-ERROR:</b> The device stopped the communication to all slaves and reached the auto-clear end state
MSK_PROFIBUS_APM_EXT_STA_NEXC_ERR	<b>NON-EXCHANGE-ERROR:</b> At least one slave has not reached the data exchange state and no process data are exchanged with it.
MSK_PROFIBUS_APM_EXT_STA_FATL_ERR	<b>FATAL-ERROR:</b> Because of a severe bus error, no bus communication is possible any more
MSK_PROFIBUS_APM_EXT_STA_NRDY	<b>Host-NOT-READY-NOTIFICATION:</b> 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
MSK_PROFIBUS_APM_EXT_STA_TOUT	<b>TIMEOUT-ERROR:</b> 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.

Table 15: Error Types in Global Bits

#### ■ 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 6.1.3 "PROFIBUS\_FSPMM\_CMD\_SET\_MODE\_REQ/CNF – Set a new Operation Mode" of this document. Allowed operation modes are:

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

Table 16: 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, also see section 0 for more information. If you want to change this state, you 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 6.1.3 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 area

This data block is reserved.

- `abSl_cfg[16]`/ Slave configuration area

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:

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x60	7	6	5	4	3	2	1	0
0x61	15	14	13	12	11	10	9	8
0x62	23	22	21	20	19	18	17	16
...								
0x6F	127	126	125	124	123	122	121	120

Table 17: 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]`/ Slave state information area

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:

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x70	7	6	5	4	3	2	1	0
0x71	15	14	13	12	11	10	9	8
0x72	23	22	21	20	19	18	17	16
...								
0x7F	127	126	125	124	123	122	121	120

Table 18: 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 data.

■ `abSl_diag[16]`/ Slave diagnostic area

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`.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x80	7	6	5	4	3	2	1	0
0x81	15	14	13	12	11	10	9	8
0x82	23	22	21	20	19	18	17	16
...								
0x8F	127	126	125	124	123	122	121	120

Table 19: 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	<ul style="list-style-type: none"> <li>■ No DataIOExchange between master and slave.</li> <li>■ Perhaps this slave is not configured or not able to respond.</li> </ul>	<ul style="list-style-type: none"> <li>■ Slave is present on the bus.</li> <li>■ DataIOExchange between master and slave.</li> </ul>
abSl_diag = 1	<ul style="list-style-type: none"> <li>■ The master and the corresponding slave do not exchange their I/O data.</li> <li>■ The master holds newly received diagnostic data in the internal diagnostic buffer.</li> </ul>	<ul style="list-style-type: none"> <li>■ Slave is present on the bus.</li> <li>■ The master and the corresponding slave exchange their I/O data.</li> <li>■ The master holds newly received diagnostic data in the internal diagnostic buffer.</li> </ul>

Table 20: Relationship between *abSl\_state* and *abSl\_diag* bits

The following error numbers are valid for *bErr\_Event*, if *bErr\_Rem\_Adr* is not equal to 255:

<i>bErr_Event</i>	signification	error source	help
2	station reports overflow	Master telegram	check length of configured slave configuration or parameter data.
3	request function of master is not activated in the station	Master telegram	check slave whether actually compatible to PROFIBUS-DP norm
9	no answer-data, although the slave must response with data	Slave	check configuration data of the station and compare it with the physical I/O data length
17	no response of the station	Slave	check bus cable, check the bus address of slave
18	master not into the logical token ring	Device	check FDL-Address of master or highest-station-Address of other master systems. examine bus cabling to detect possible bus short circuits.
21	incorrect parameter in request	master telegram	contact Hilscher's hotline

Table 21: Errors which may occur in the Network (*bErr\_Rem\_Adr* is not equal to 255)

The following error numbers are valid for bErr\_Event, if bErr\_Rem\_Adr is equal to 255:

bErr_Event	signification	error source	help
0	no mistakes appear		
50	AP-Task not found	Device	contact technical support
51	no global data-field	Device	contact technical support
52	DL-Task not found	Device	contact technical support
53	FSPMM-Task not found	Device	contact technical support
54	non existing master parameters	Device	execute download of data base again
55	incorrect parameter-value in the master parameters	project planning	contact technical support
56	non existing slave parameters	project planning	execute download of data base again
57	incorrect parameter-value in a slave parameters data file	project planning	contact technical support
58	double slave address	project planning	check projected addresses
59	projected send process data offset address of a participant outside the allowable border of 0- 255	project planning	check projected addresses
60	projected receive process data offset address of a participant outside the allowable border of 0- 255	project planning	check projected addresses
61	data-areas of slaves are overlapping in the send process data	project planning	check projected addresses
62	data-areas of slaves overlapping in the receive process data	project planning	check projected addresses
63	unknown process data handshake	warmstart	check warmstart parameters
64	free RAM exceeded	Device	contact technical support
65	incorrect slave parameter data sets	project planning	contact technical support
202	no segment for the treatment free	Device	contact technical support
212	incorrect reading of a data base	Device	execute download of data base again
213	structure-surrender to operating system faulty	Device	contact technical support
220	Software Watchdog error	Host	check Host program

bErr_Event	signification	error source	help
221	No Data Acknowledge in process data handshake mode 0	Host	Host program hasn't acknowledge the last handshake in time
222	Master in Auto_Clear	Slave Device	the auto_clear mode was activated, because one slave is missing during runtime
225	No further Segments	Device	contact technical support

Table 22: Errors which may occur in the Profibus DP Master Device (bErr\_Rem\_Adr is equal to 255)

Besides the global bits, master main state, error informations and counters, additional status bit lists of slaves are defined in this structure (namely slave configuration area, slave state information area, slave diagnostic area). These bit lists contain the current state information of all slave devices the master communicates with (i.e. 16 bytes = 128 devices). Despite the fact that the implementation of extended status block is protocol specific, the place and definition of these bit lists are to a greater or lesser extent similar for all Hilscher Fieldbus Master protocol stacks. The layout of this block is still maintained with actual specification and will be supported further. The example below shows a generic way to define the corresponding location of the bitlists located at the offsets 0x60, 0x70 and 0x80 (see Table above). Three state structures are needed to be defined to locate such bitlists i.e. inside of input data block.

Extended Status Block for PROFIBUS DP-Master – Second part (State Field Definition Block)			
Offset	Type	Name	Description/Value
0x00FC	unsigned char	bReserved[3]	Reserved. Do not use.
0x00FF	unsigned char	bNumStateStructs	Number of State Structures defined below = 3
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[0]	<b>Structure to define State field properties</b> <b>For</b>
0x0100	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=1. Corresponds to a bitlist (one bit per node ) of configured nodes
	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 configuration area. See <a href="#">description of the slave configuration area above</a> .
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[1]	<b>Structure to define State field properties</b>
0x0110	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=2. Corresponds to a bitlist (one bit per node ) of active nodes

	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. See <a href="#">description of the slave state information area above</a> .
↓	<b>NETX_EXTENDED_STATE_STRUCT_T</b>	<b>atStateStruct[2]</b>	<b>Structure to define State field properties</b>
<b>0x0120</b>	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=3. Corresponds to a bitlist (one bit per node ) of diagnostic nodes
	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. See <a href="#">description of the slave diagnostic above</a> .

Table 23: Extended Status Block for PROFIBUS DP-Master – Second part (State Field Definition Block)

If the location of the state fields is defined to be inside of input data area 0 block (as it is shown in generic example above), the corresponding bitlists will be updated by the stack consistently to the data in this area. Moreover, the data and corresponding state fields can be read out by the host application as one data block i.e. with DMA support.

### 3.4 Control Block

A control block is always present in both system and communication channel. In some respects, control and status block are used together in order to exchange information between host application and netX firmware. The control block is written by the application, whereas the application reads a status block. Both control and status block have registers that use the *Change of State* mechanism (see section 0).

The following gives an example of the use of control and status block. The host application wishes to lock the configuration settings of a communication channel to protect them against changes. The application sets the *Lock Configuration* flag in the control block to the communication channel firmware. As a result, the channel firmware sets the *Configuration Locked* flag in the status block (see below), indicating that the current configuration settings cannot be deleted, altered, overwritten or otherwise changed.

The control block of a dual-port memory features a watchdog function to allow the operating system running on the netX supervise the host application and vice versa. The control area is always present in the dual-port memory.

Control Block			
Offset	Type	Name	Description
0x0008	UINT32	ulApplicationCOS	Application Change Of State State Of The Application Program INITIALIZATION, LOCK CONFIGURATION
0x000C	UINT32	ulDeviceWatchdog	Device Watchdog Host System Writes, Protocol Stack Reads

Table 21 - Communication Control Block

#### Communication Control Block Structure

```
typedef struct NETX_CONTROL_BLOCK_Ttag
{
  UINT32 ulApplicationCOS;
  UINT32 ulDeviceWatchdog;
} NETX_CONTROL_BLOCK_T;
```

For more information concerning the Control Block please refer to the netX DPM Interface Manual.

## 4 Getting started / Configuration

This section explains some essential information you should know when starting to work with the PROFIBUS-Master Protocol API.

### 4.1 Overview about Essential Functionality

You can find the most commonly used functionality of the PROFIBUS-Master Protocol API within the following sections of this document:

Topic	Section Number	Section Name
Cyclic data transfer (Input/Output)	6.1.8	PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input
	6.1.7	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output
Acyclic data transfer (Records)	6.1.9	PROFIBUS_FSPMM_CMD_READ_REQ/CNF –V1 Class 1 Read
	6.1.10	PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – V1 Class 1 Write Request
Alarm management	6.1.11	PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification
	6.1.12	PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge

*Table 24: Overview about Essential Functionality (Cyclic and acyclic Data Transfer and Alarm Handling).*

## 4.2 Configuration of Bus and Slave Parameters

This section explains how to configure the bus, the master and the slaves correctly. This can be done either by sending packets writing the parameters at the correct places in the dual-port memory or by using Hilscher's configuration tool SyCon.net. The section introduces both methods and contains a detailed description of all parameters.

### 4.2.1 Write Access to the Dual-Port Memory

In order to change the bus parameters for the master in the dual-port memory, a PROFIBUS\_FSPMM\_CMD\_INIT\_REQ/CNF – Initialization Command packet has to be sent to the protocol stack. For more information how to accomplish this, please refer to [section 6.1.1 of this manual](#)

Similarly, in order to configure the slaves and to supply them with warmstart parameters, the PROFIBUS\_FSPMM\_CMD\_DOWNLOAD\_REQ/CNF – Download Slave Parameter Set has to be sent to the protocol stack. More information about this topic can be obtained at [section 6.1.13 of this manual](#).

### 4.2.2 Using the configuration tool SYCON.net

The configuration tool SYCON.net is described in an own manual.

### 4.2.3 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:

Parameter	Meaning	Range of Value / Value
Bus_Para_Len	Contains the length of the Bus_Para including the field Bus_Para_Len itself	34
FDL_Add	Contains the own address of the PROFIBUS Device	1-125
Baud_rate	Contains the transmission speed	0-11
T <sub>SL</sub>	Slot-time	37-16383
Min T <sub>SDR</sub>	Min. station delay responder	1-1023
Max T <sub>SDR</sub>	Max. station delay responder	1-1023
T <sub>QUI</sub>	Quiet time	0-127
T <sub>SET</sub>	Setup time	1-255
T <sub>TR</sub>	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 USER Master data	0-32
Master_Class2_Name[32]	Name of the Master Class 2 (currently not used)	
Master_User_Data[32]	USER specific Parameter data	0-32
T <sub>CL</sub>	Isochronous cycle time	
Max_T <sub>SH</sub>	Maximum Shift Time	

Table 25: Bus and Master Parameters, their Meanings and their Ranges of allowed Values

#### 4.2.3.1 Bus\_Para\_Len

This parameter is calculated as the length of the bus parameter structure in bytes including the field itself.

#### 4.2.3.2 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.

#### 4.2.3.3 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 kBits/s	0
PROFIBUS_DL_DATA_RATE_19_2	19,2 kBits/s	1
PROFIBUS_DL_DATA_RATE_93_75	93,75 kBits/s	2
PROFIBUS_DL_DATA_RATE_187_5	187,5 kBits/s	3
PROFIBUS_DL_DATA_RATE_500	500 kBits/s	4
PROFIBUS_DL_DATA_RATE_1500	1500 kBits/s	6
PROFIBUS_DL_DATA_RATE_3000	3000 kBits/s	7
PROFIBUS_DL_DATA_RATE_6000	6000 kBits/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 26: Supported Baud Rates

#### 4.2.3.4 $T_{SL}$ (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 baudrate.

#### 4.2.3.5 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.

#### 4.2.3.6 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.)

#### 4.2.3.7 $T_{QUI}$ (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.)

#### 4.2.3.8 $T_{SET}$ (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.)

#### 4.2.3.9 $T_{TR}$ (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)

#### 4.2.3.10 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  $G \cdot TTR$ , 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.)

#### 4.2.3.11 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.

#### 4.2.3.12 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.

#### 4.2.3.13 Bp\_Flag

This flag decides on the system behaviour 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.

#### 4.2.3.14 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.

#### 4.2.3.15 Poll\_Timeout

The Class2 Poll timeout parameter is only relevant when working as DP V1 Class 2 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).

#### 4.2.3.16 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 (class 1) 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.

#### 4.2.3.17 Alarm\_Max

This parameter specifies the allowed maximum number of alarms that can be handled in parallel by the master. This is a capability of the Profibus DP master. As there are seven different alarm classes, this value must be at least 7.

#### 4.2.3.18 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.

#### 4.2.3.19 Reserved

This is a reserved area.

#### 4.2.3.20 Master\_User\_Data\_Len

It specifies the length of the user data area at the master. It may not exceed 32, see Master\_User\_Data[32] below.

#### 4.2.3.21 Master\_Class2\_Name[32]

The 'Master Class2 Name' parameter is only relevant when working as DP V1 Class 2 master (currently not supported). It represents the name of a Profibus DP V1 Class 2 master, if any is present within the Profibus DP network. It is a field with a length of 32 bits.

#### 4.2.3.22 Master\_User\_Data[32]

The first two bytes are used to set the PROFIBUS Ident Number. Therefore the variable Master\_User\_Data\_Len has to be set to 2. If the values are set to 0 the stack will ignore this ident number.

```
Master_User_Data[0] = IdentNumberLowByte;  
Master_User_Data[1] = IdentNumberHigh Byte
```

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[2] = bHardVrsUsr  
Master_User_Data[3] = bFirmVrsUsr
```

These variables are returned with service Get\_Master\_Diag() of a master - master communication. Usually these values can be set to 0 and the stack will use its internal values. If it is necessary to change these values e.g. to adopt to your own HW revision, they can be overwritten if the particular value is <> 0. Therefore the variable Master\_User\_Data\_Len has to be set to 6.

**4.2.3.23  $T_{CL}$** 

Isochronous cycle time.

---

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

---

**4.2.3.24  $Max\_T_{SH}$** 

Maximum shift time.

---

**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:

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

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

$$T_{QUI} < T_{RDY}$$

## 4.2.4 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 data	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 27: Slave Parameters, their Meanings and their Ranges of allowed Values

#### 4.2.4.1 Slave Address

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

#### 4.2.4.2 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.

#### 4.2.4.3 SI\_Flag

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

Explanation of Bits of <code>s1_Flag</code> 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 according to IEC 61158/EN 50170
		1	CFG data according to DPV1
D1	Extra_Alarm_SAP	0	DPV1-Master acknowledges alarms via SAP 51
		1	DPV1-Master acknowledges alarms via SAP 50
D0	Reserved		Reserved

Table 28: Explanation of Bits of `SI_Flag` within Slave Parameter Block

#### 4.2.4.4 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 29: Meaning of Slave\_Type

#### 4.2.4.5 Max\_Diag\_Data\_Len

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

#### 4.2.4.6 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

- $Max\_Diag\_Data\_Len - 6$
- and 64.

#### 4.2.4.7 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).

#### 4.2.4.8 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).

#### 4.2.4.9 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 30: Meaning of Alarm\_Mode Parameter

#### 4.2.4.10 Add\_SI\_Flag

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

Explanation of Bits of Add_SI_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 31: Explanation of Bits of SI\_Flag within Slave Parameter Block

#### 4.2.4.11 C1\_Timeout

This parameter indicates a 16-bit value specifying a timeout value for Class 1 services.

#### 4.2.4.12 bReserved[4]

This represents a reserved area.

#### 4.2.4.13 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

#### 4.2.4.14 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	<u>Watchdog control</u>	
		0	Watchdog control is deactivated
		1	Watchdog control is activated
D2-D0	Reserved		Reserved for future extensions

Table 32: 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 33: 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 paramters**

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.

**4.2.4.15 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

**4.2.4.16 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.

**4.2.4.17 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

#### 4.2.4.18 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
<code>Input_Count</code>	byte	number of input offsets following in the <code>IO_Offset</code> table
<code>Output_count</code>	byte	number of output offsets following in the <code>IO_Offset</code> table
<code>IO_Offsets[...]</code>	word array	word or byte <code>IO_Offset</code> in the order: first all input offsets then all output offsets in case of modular stations

Table 34: `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. 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.

#### 4.2.4.19 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.

#### 4.2.4.20 Slave\_User\_Data

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

### 4.3 Task Structure of the PROFIBUS DP-Master Stack

The figure below displays the internal structure of the tasks which together represent the PROFIBUS DP-Master Stack:

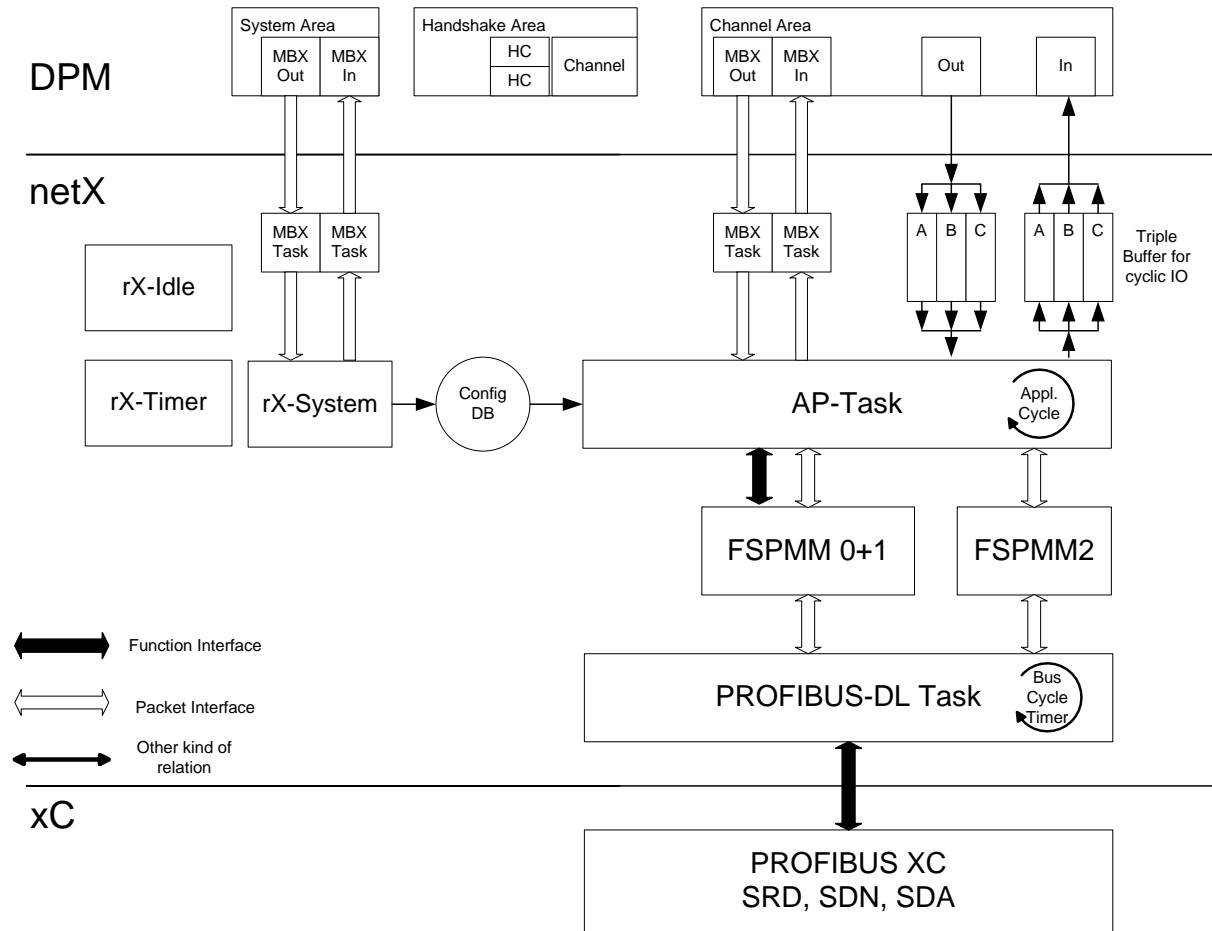


Figure 6: Internal Structure of PROFIBUS DP-Master Firmware

For the explanation of the different kinds of arrows see lower left corner of figure.

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 DualPort 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 Class 1 functionality. In detail, it handles the following items:

- Cyclic Communication
- Acyclic Communication DP V1 Class 1
- Alarm handling

### **PROFIBUS FSPMM2 task**

The PROFIBUS FSPMM2 task is responsible for the Profibus V1 Class 2 functionality. In Detail, it handles the following items:

- Acyclic Communication DP V1 Class 1
- Master 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)

### 4.3.1 Obtaining Diagnostic Information from connected Slaves by sending an RCX\_GET\_SLAVE\_CONN\_INFO\_REQ Packet

An application which is based on Hilscher's DPM for netX can obtain diagnostic and status information about all slaves connected to and administered by this master from the master firmware as described in general in the netX DPM Manual (Ref. 2). (This information only concerns cyclic data transfer.) The PROFIBUS DP Master firmware supports this feature.

- The netX operating system rcX uses handles in order to access at the slaves.

Retrieving the diagnostic information is a two-step-process as you first retrieve the handle using the *Get Slave Handle* request and subsequently you retrieve the diagnostic information using the handle.

1. Retrieve the handle by the *Get Slave Handle* request (RCX\_GET\_SLAVE\_HANDLE\_REQ, Command code 0x2F08) which is described in the netX DPM Manual (Ref. 1), chapter 5.2.2.1. In order to do so, you need to choose which kind of list of the above mentioned slave lists you want to obtain. The confirmation packet you will receive (RCX\_GET\_SLAVE\_HANDLE\_CNF, Command code 0x2F09) will then deliver an array of handles to the elements of the selected list.

This allows to obtain a diagnosis structure for the specific slave by the *Get Slave Connection Information* request (RCX\_GET\_SLAVE\_CONN\_INFO\_REQ, Command code 0x2F0A). This packet requires the handle of the specific slave taken from the array of handles to the elements of the selected list obtained in the first step. You will then receive the confirmation packet (RCX\_GET\_SLAVE\_CONN\_INFO\_CNF, Command code 0x2F0B) delivering – besides others – a structure `tState` containing the following structure with information about the selected slave from the master firmware (see reference 1 for more details). For an PROFIBUS DP Slave this structure (PROFIBUS\_FSPMM\_DIAGNOSTIC\_DATA\_T) structure looks like:

```
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;
```

This structure is exactly the same as the one delivered by the PROFIBUS\_FSPMM\_PACKET\_GET\_SLAVE\_DIAG\_CNF\_T confirmation packet, also see section 6.1.5, „PROFIBUS\_FSPMM\_CMD\_GET\_SLAVE\_DIAG\_REQ/CNF – Request a Slave Diagnostic“ on page 121 of this document.

The variables of this structure are explained in more detail in section 5.3.5.1 *“Diagnostic model of Profibus DP V0”* (description of octets 1 to 3):

## 5 Overview

### 5.1 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)

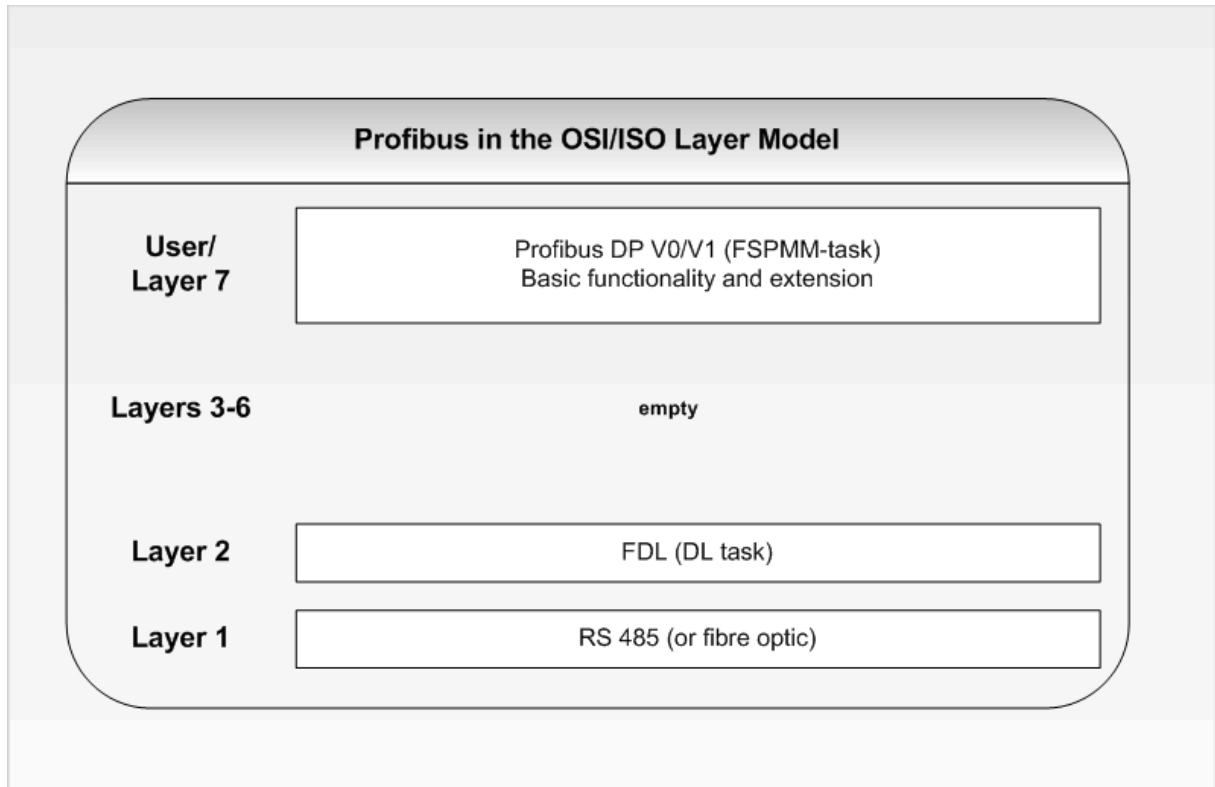


Figure 7: Profibus in the OSI/ISO Layer Model

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.

## 5.2 Profibus DP Operation Modes (States)

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

- OFFLINE (USIF\_OFFLINE)
- STOP (USIF\_STOP)
- CLEAR (USIF\_CLEAR)
- OPERATE (USIF\_OPERATE)

These states differ in the degree of allowed functionality as follows:

- 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.
- 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.
- 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.
- 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.

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

The system behaviour 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", see section 0.

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 of this document.

## 5.3 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 Class 1
- Configuration of Inputs and Outputs at Slaves
- Alarm Processing
- Diagnostics

### 5.3.1 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. The main packets for cyclic data transfer are:

#### Packets for Cyclic Data Transfer

Section	Packet Name
6.1.8	PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input
6.1.7	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output

Table 35: Packets for Cyclic Data Transfer

#### 5.3.1.1 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 4.2.4 Detailed Description of Slave Parameters, especially 4.2.4.15 `Cfg_Data_Len` and 4.2.4.16 `Cfg_Data`
- Section 6.1.1 `PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register`
- Section 6.3.14 `PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set`

### 5.3.1.2 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 4.2.3.13 "`Bp_Flag`" of this document.

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.

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

### 5.3.1.4 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 a '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 6.1.14 "`PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF` – Global Control Message".

## 5.3.2 Acyclic Data Transfer

### 5.3.2.1 Acyclic Data Transfer of the DP Master Class 1

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

In Profibus 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.

The main packets for acyclic data transfer are:

Section	Packet Name
6.1.9	PROFIBUS_FSPMM_CMD_READ_REQ/CNF –V1 Class 1 Read
6.1.10	PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – V1 Class 1 Write Request

*Table 36: Packets for Acyclic Data Transfer*

With these packets, you can access data areas within single modules of the slave. 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 desired 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 5.3.2.2 of this document.

**5.3.2.2 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 Class 1 and DP V1 Class 2. 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 37: Explanation of Error Class and Error Code within Error\_Code\_1

Error\_Code\_2 is fully user specific and cannot be explained in more detail therefore .

### 5.3.3 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 slaves 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.

#### 5.3.3.1 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.

### 5.3.3.2 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 38: 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.

### 5.3.3.3 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 39: Special Identifier Format of Identifier Byte according to IEC 61158/EN 50170 Specification

### 5.3.3.4 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 40: Structure of Length Byte in the Special Identifier Format of the Identifier Byte according to IEC 61158/EN 50170 Specification

## 5.3.4 Alarm Processing

### 5.3.4.1 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 explicite acknowledgement. The reception of the alarm message from the slave causes a PROFIBUS\_FSPMM\_CMD\_ALARM\_NOTIFICATION\_IND indication at the master.

The aforementioned explicite 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 a confirmation message (PROFIBUS\_FSPMM\_CMD\_ALARM\_ACK\_CNF).

The importance of this explicite 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 Class 1 masters.

### 5.3.4.2 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 such as 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 operator 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)

### 5.3.4.3 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 defined either as a limit of totally allowed alarms independently from the alarm type or there is only one alarm of each specified type permitted.

### 5.3.4.4 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 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 6.1.11 “*PROFIBUS\_FSPMM\_CMD\_ALARM\_NOTIFICATION\_IND – Alarm Notification*” of this document, see *Table 77: Available 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 5.3.5 “*Diagnosis*”.

For more information, please refer to

- Section 6.1.11 “*PROFIBUS\_FSPMM\_CMD\_ALARM\_NOTIFICATION\_IND – Alarm Notification*”.
- Section 6.1.12 “*PROFIBUS\_FSPMM\_CMD\_ALARM\_ACK\_REQ/CNF – Alarm Acknowledge*”

### 5.3.5 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.

### 5.3.5.1 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

D7	D6	D5	D4	D3	D2	D1	D0	Description
							x	Station_Non_Existent
						x		Station_Not_Ready
					x			Cfg_Fault
				x				Ext_Diag
			x					Not_Supported
		x						Invalid_Slave_Response
	x							Prm_Fault
x								Master_Lock

Table 41: Octet 1: Station Status\_1

These bits have the following meaning:

- **Station\_Non\_Existent**

This bit indicates that the slave does not exist or is not responding. An operating slave always sets this bit to 0.

- **Station\_Not\_Ready**

This bit indicates that the slave is not ready for cyclic data transfer.

- **Cfg\_Fault**

This bit 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.

- **Ext\_Diag**

This bit 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 5.3.5.2 "Diagnostic model of Profibus DP V1".

- **Not\_Supported**

This bit indicates that an unknown command has been detected by the slave. The requested command is not supported by the slave.

- **Invalid\_Slave\_Response**

This bit indicates that the response of the slave was invalid or not plausible. An operating slave always sets this bit to 0.

- **Prm\_Fault**

This bit indicates that the last received parameter telegram was defective or incorrect.

- **Master\_Lock**

This bit 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.

#### Octet 2: Station Status\_2

D7	D6	D5	D4	D3	D2	D1	D0	Description
							x	Prm_Req
						x		Stat_Diag
					x			1
				x				WD_On
			x					Freeze_Mode
		x						Sync_Mode
	x							Reserved
x								Deactivated

Table 42: Octet 2: Station Status\_2

The bits of octet 2 have the following meaning:

- **Prm\_Req**

This bit 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'.

- **Stat\_Diag**

This bit (statistical diagnosis) 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.

- **Fixed 1**

This bit is always set to the value '1' indicating DP V1 operation.

- **WD\_On**

This bit indicates that the watchdog timer supervision mechanism for the slaves is activated.

- **Freeze\_Mode**

This bit indicates that the slave has received a 'Freeze' command and since then no 'Unfreeze' command.

- **Sync\_Mode**

This bit indicates that the slave has received a 'Sync' command and since then no 'Unsync' command.

- **Reserved**

This bit is reserved by the DP standard.

- **Deactivated**

The slave has not been projected. A projected slave always sets this bit to 0.

#### Octet 3: Station Status\_3

D7	D6	D5	D4	D3	D2	D1	D0	Description
	x	x	x	x	x	x	x	Reserved
x								Ext-Diag_Overflow

Table 43: Octet 3: Station Status\_3

The bits of octet 3 have the following meaning:

- **Reserved**

These bits are reserved.

- **Stat\_Diag**

This bit indicates that the slave has more diagnostic data available than it can send

#### Octet 4: Master\_Add

This byte contains the master address from 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.

### 5.3.5.2 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 5.3.4.4 “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:

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. The length of one block is limited to 64 bytes including the header, the length of all blocks together to 238 bytes.

The first byte of each block is header byte. Its first two most significant bytes code the type of diagnostic block. The rest of contains the length of the alarm message in bytes (up to 63 bytes are available here).

**Device-related diagnosis**

In DP V0, interpretation of device-related diagnostic information depends on the contents of the \*.GSD file of the device. The header has the standard structure:

D7	D6	D5	D4	D3	D2	D1	D0	Description
0	0							Indicating device-related diagnosis
		0-63						Length of diagnostic block (in bytes)

Table 44: Header of Device-related Diagnosis

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 header has the following structure:

D7	D6	D5	D4	D3	D2	D1	D0	Description
0	1							Indicating identification-related diagnosis
		0-63						Identifier_Number of related module

Table 45: Header of Identification-related Diagnosis

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 46: Identification-related diagnosis – Second Byte

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 47: Identification-related diagnosis – Third 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 than the identification-related diagnosis and reports about channel errors within the modules of a slave.

For each channel an own diagnostic block with a length of 3 bytes is used.

For channel -related diagnosis, the header byte has the following structure:

D7	D6	D5	D4	D3	D2	D1	D0	Description
0	1							Indicating channel -related diagnosis
		0-63						Length of diagnostic block (in bytes)

Table 48: Header of Channel-related Diagnosis

The second byte contains the following information

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

Table 49: Channel-related Diagnosis – Second 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						Bit
0	1	0						2 Bits
0	1	1						4 Bits
1	0	1						Byte
1	1	0						Word
1	1	1						2 Words
			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 50: Channel-related Diagnosis – Third Byte

## 5.4 Functionality of the FSPMM2-Task

This task provides the functionality of DP V1 Class 2 on layer 7 of the OSI/ISO layer model. The acyclic data processing capabilities for Profibus DP Masters Class 2 provided by the FSPMM2-task are described in this section.

The error handling is the same as already described for DP V1 Class 1, so please refer to section 5.3.2.2 “**DP V1 Error Processing**” above.

### 5.4.1 General Remarks on DP V1 Class 2

The DP V1 Class 2 technology is characterized by the following features:

- Connection-oriented data transmission over a DP V1Class 2 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’).
- A master class 2 can only have one connection to a slave at a time.
- A slave may have multiple connections to a master class 2 and one to a master class 1 at a time.

### 5.4.2 DP V1 Class 2 Masters

Class 2 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

Class 2 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.

Class 2 masters acyclically communicate with slaves over a class 2 connection (also called MSAC\_C2 connection relationship). This class 2 connection must explicitly be set up.

### 5.4.3 Basic Services for Connection Maintenance

Two services are defined for handling a class 2 connection between a class 2 master and a slave:

- MSAC2\_Initiate service.
  - This service is used by the DP master class 2 in order to set up a DP V1-Class 2 connection to the DP slave. The DP slave then receives a “Initiate” indication.
- MSAC2\_Abort service.
  - This service is used by the DP master class 2 in order to abort a DP V1-Class 2 connection to the DP slave. The DP slave then receives an abort indication. The DP slave may also request an abort of the class 2 connection by itself. For instance, this can be accomplished by sending the abort packet.

#### 5.4.4 Basic Services available after Establishing a DP V1-Class 2 Connection

The following services are available after a DP V1-Class 2 connection has been set up successfully:

- *DS\_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.
- *DS\_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.
- *DS\_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.

Only one of these acyclic services can be executed at a time per MSAC\_C2 connection

#### 5.4.5 Overview of Supported Functionality

The packets listed below are available for providing these acyclic services of DP V1 Masters Class 2. They are described in section 6.2 “*The FSPMM2-Task*” of this document.

##### Supported Functionality of Profibus DP V1 Class 2 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 - V1 Class 2 Read Request
6.2.4	PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class 2 Write Request
6.2.5	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined V1 Class 2 Read and Write Request
6.2.6	PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection
6.2.7	PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ/CNF – Read Slave Diagnostics (DP V1 Class 2)
6.2.8	PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ/CNF – Read Input Values
6.2.9	PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ/CNF – Read Output Values
6.2.10	PROFIBUS_FSPMM2_CMD_GET_CFG_REQ/CNF – Get Configuration Command
6.2.11	PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ/CNF – Set Slave Address (DP V1 Class 2)
6.2.12	PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ/CNF – Get Master Diagnosis
6.2.13	PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ/CNF – Live List Command
6.2.14	PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication
6.2.15	PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication

Table 51: Overview of available Packets supporting DP V1 Class 2

### 5.4.6 Extended Addressing Mechanism

Principally, DP V1-Class 2 uses the same addressing mechanism as DP V1-Class 1 described above: All data blocks (length 244 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 class 2 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 class 1. Which API is used depends on the choice made during initialization of the MSAC\_C2 connection, so this choice determines the API for the whole lifetime of the connection.

### 5.4.7 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 Class 2 are available.

However, at one time only one connection may be held by a DP V1 class 2 master, where as one slave may be connected to several DP V1 class 2 masters and a class 1 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.

### 5.4.8 Initialization Process of a DP V1-Class 2 Connection

The establishment of an MSAC\_C2 connection is a process with several stages:

- 1) First, the Profibus DP Master class 2 sends a special request (Initiate-REQ-PDU) to a special service access point of the Profibus DP Slave.
- 2) 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 class 2.
- 3) At the same time the 'Initiate' indication is sent from the Profibus DP Slave to the slave application.
- 4) The slave now sends a message (RM-REQ-PDU) to the master.
- 5) This message is received at the master. The service access point will be stored then.
- 6) 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.
- 7) The master now waits for the positive or negative reaction of the slave in a polling mode.
- 8) The slave will react on reception of the response to the 'Initiate' indication.
- 9) 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.

### Initialization Sequence of DP V1 Class 2-Connection

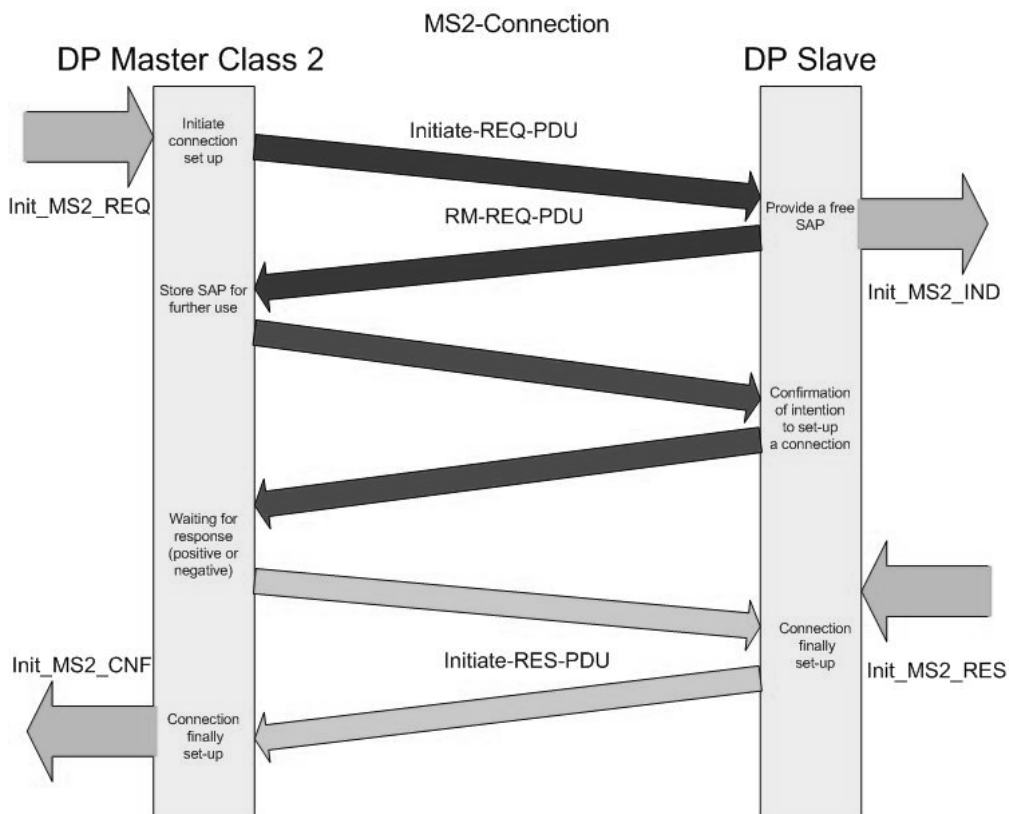


Figure 8: Initialization Sequence of DP V1 Class 2-Connection

## 5.4.9 Important Parameters

The following parameters are relevant when establishing an MSAC\_C2 connection.

### 5.4.9.1 Features supported

DP master class 2 and DP slave communicate about the supported functionality of each other. This gives the slave the opportunity to adjust its functionality to the masters requirement or to reject the request if it cannot fulfill them.

### 5.4.9.2 Profile Features supported

DP master class 2 and DP slave communicate about the supported profile features. The meaning of the bits of these two bytes depends on the profile or vendor.

### 5.4.9.3 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.

### 5.4.9.4 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

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.

### 5.4.10 Requirements for the Class 2 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 an `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.

### 5.4.11 Connection Supervision by Internal Timers

The once established DP V1 Class 2 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 class 2 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 Class 2 connection are:

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

## 5.5 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_SENDFIFO/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

### 5.5.1 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.

#### 5.5.1.1 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).

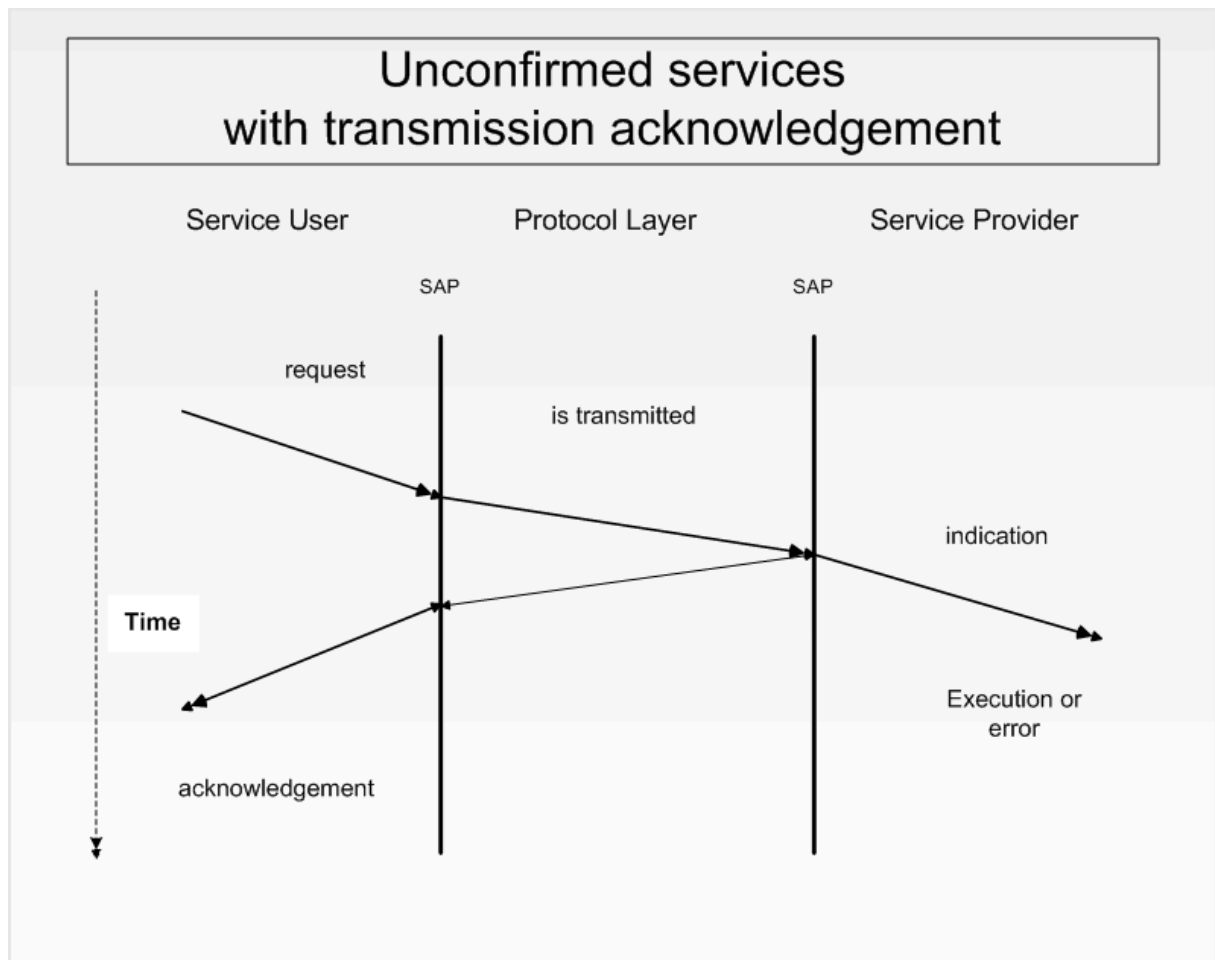


Figure 9: SDA Service with Acknowledgement

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

### 5.5.1.2 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

### 5.5.1.3 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

#### 5.5.1.4 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.

## 5.5.2 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

### 5.5.2.1 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.

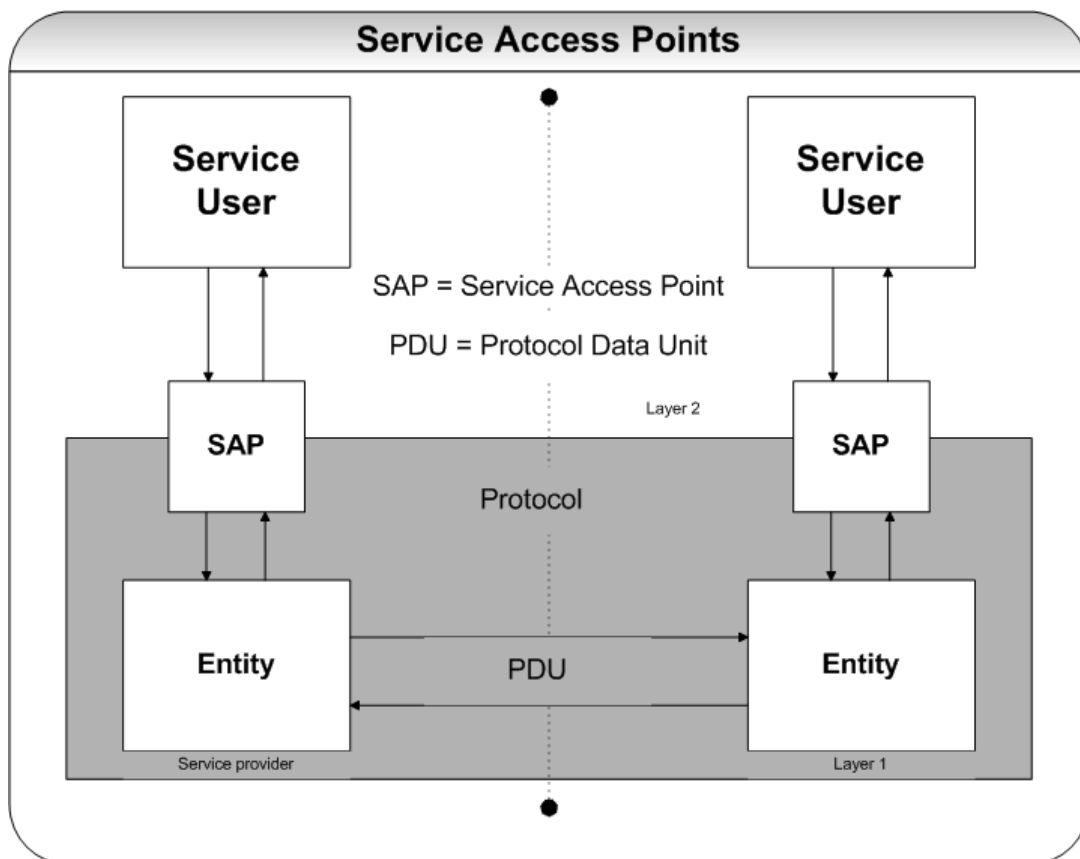


Figure 10: Service Access Points

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 SAP's 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 SAP's between user services and

management services, so you can introduce a separation between user SAP's (which might be activated by this packet) and previously activated management SAP's.

Along with the SAP's come automatic checking mechanism for rule conformity of the messages and the accompanying transmission-related informations 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 SAP's 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

#### 5.5.2.2 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).

#### 5.5.2.3 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.

#### 5.5.2.4 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

### 5.5.3 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 6.3.15 " 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 6.3.13 "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\_END\_PROCESS\_REQ/CNF– Finish the DL Task

- PROFIBUS\_DL\_CMD\_START\_DLE\_REQ/CNF – Starts the DL-Layer
- PROFIBUS\_DL\_CMD\_STOP\_DLE\_REQ/CNF – Stop DL Layer

## 5.6 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 resetting the devices. Scope of this section is the definition of the configuration during network operation state OPERATE feature for the PROFIBUS network. For more information on operation states see section “*Profibus DP Operation Modes*” of this document.

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

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

### 5.6.1 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 command (`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 an `RCX_VERIFY_DATABASE_CNF` packet 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 was successful, then the Host sends the command for activating changes.
6. Now, the new configuration will be activated with the command `RCX_ACTIVATE_DATABASE_REQ`.
  - At first, the master deactivates all slaves having been changed or deleted.
  - Then the PROFIBUS-DP Master sends the new configuration to all changed or new slaves and sets them to active state regarding to the setting of the each slave.

- o 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.

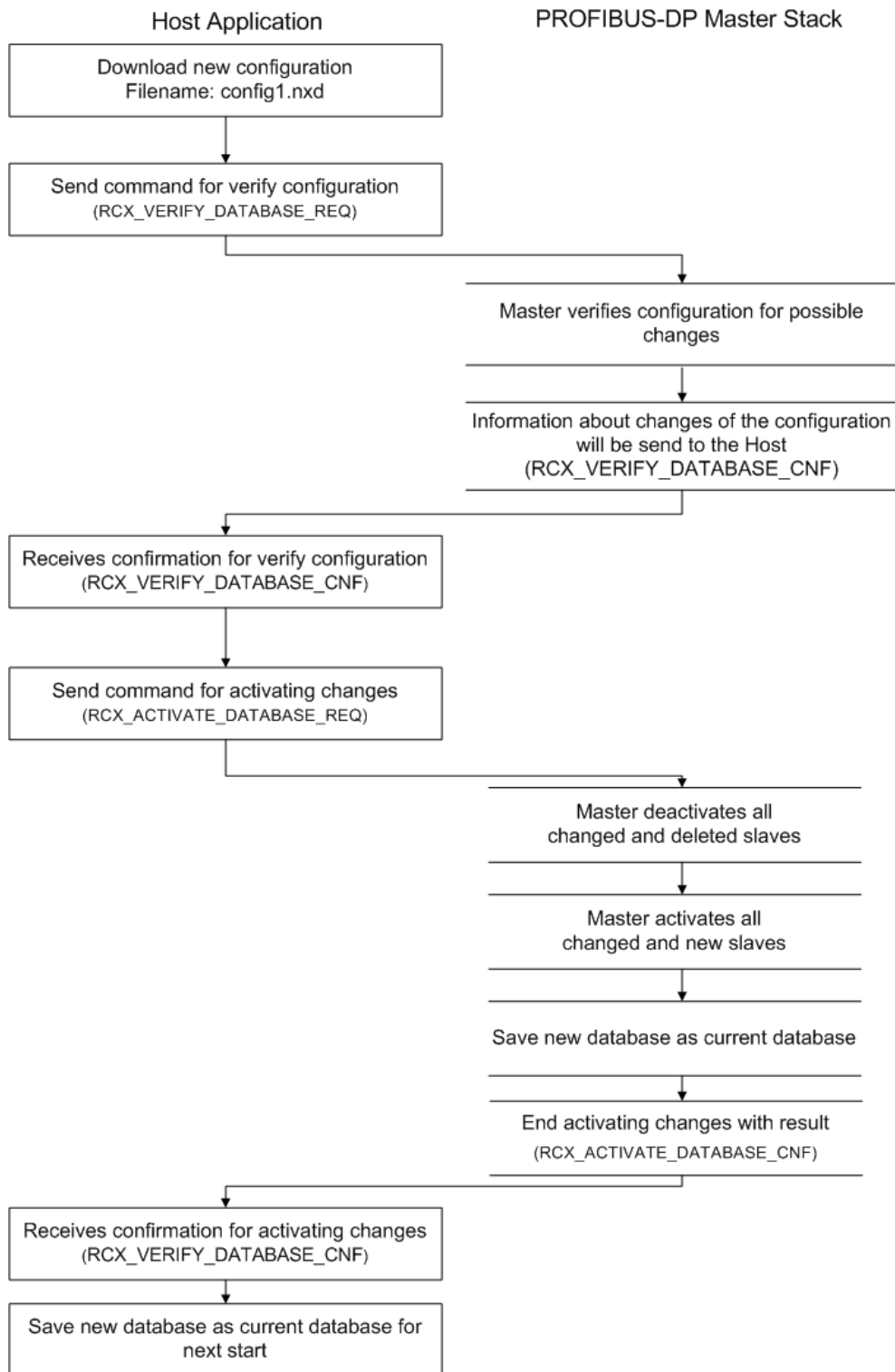


Figure 11: Configuration during network state OPERATE

## 5.7 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:** The redundancy functionality can be used with DPV0 slaves only. DPV1 slaves can fall off the bus during the switch over.

---



---

**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.

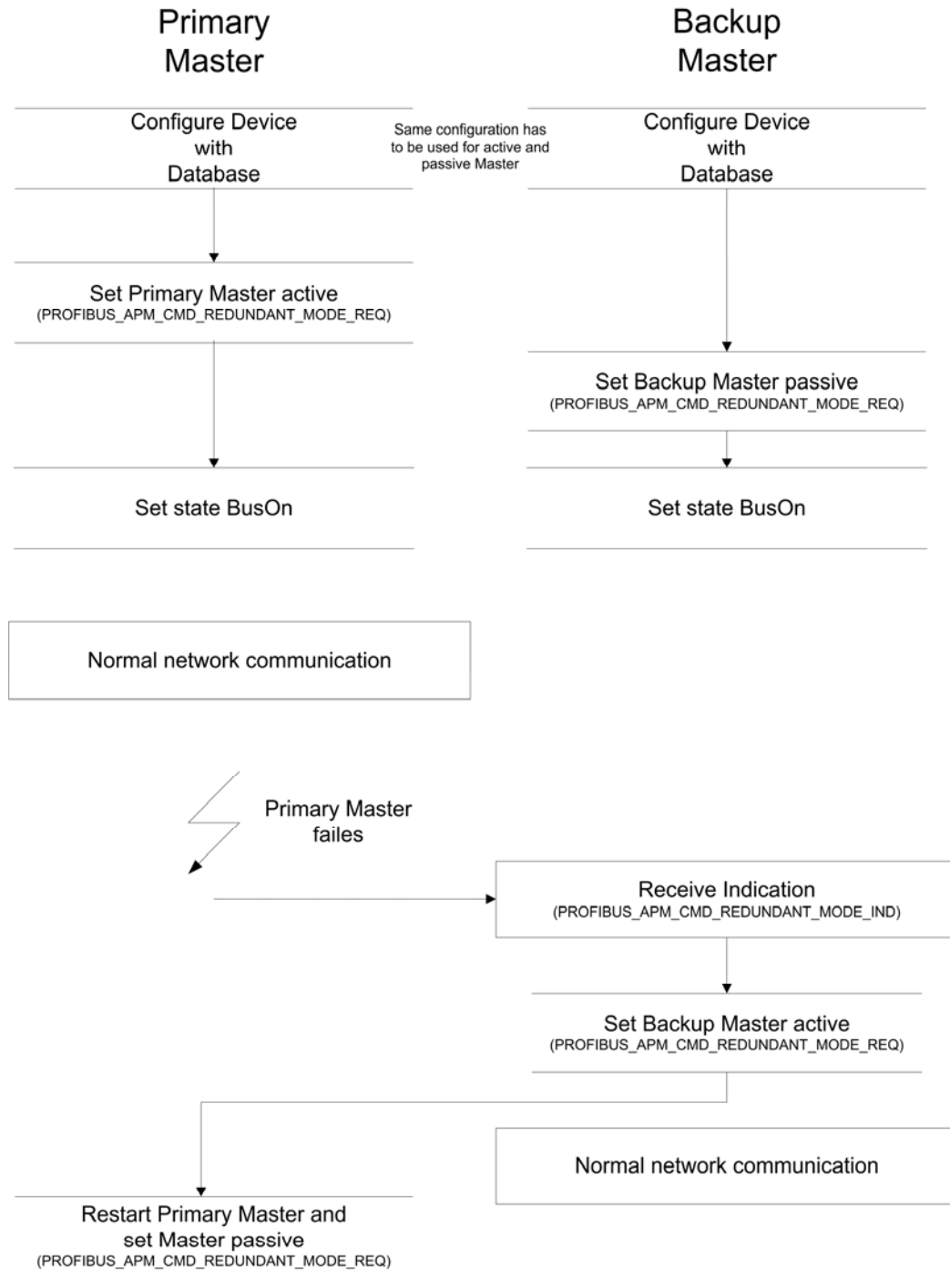


Figure 12: Redundancy Functionality – Host Application



**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.

## 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 52: 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_SENDFPACKET_FIFO/LIFO()` for sending a packet to the FSPMM-Task.

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

<b>Overview over Packets of the FSPMM -Task</b>			
<b>No. of section</b>	<b>Packet</b>	<b>Command code (REQ/CNF or IND/RES)</b>	<b>Page</b>
6.1.1	PROFIBUS_FSPMM_CMD_APP_REG_REQ/CNF – Application Register	0x2224/ 0x2225	111
6.1.2	PROFIBUS_FSPMM_CMD_INIT_REQ/CNF – Initialization Command	0x2200/ 0x2201	113
6.1.3	PROFIBUS_FSPMM_CMD_SET_MODE_REQ/CNF – Set a new Operation Mode	0x2206/ 0x2207	116
6.1.4	PROFIBUS_FSPMM_CMD_MODE_CHANGE_IND – Mode changed Indication	0x2214	119
6.1.5	PROFIBUS_FSPMM_CMD_GET_SLAVE_DIAG_REQ/CNF – Request a Slave Diagnostic	0x220A/ 0x220B	121
6.1.6	PROFIBUS_FSPMM_CMD_NEW_SLAVE_DIAG_IND - Indicate new Slave Diagnostic	0x2216	126
6.1.7	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ/CNF – Set new Output Data	0x220C/ 0x220D	127
6.1.8	PROFIBUS_FSPMM_CMD_GET_INPUT_REQ/CNF - Get new Input Data	0x220E/ 0x220F	130
6.1.9	PROFIBUS_FSPMM_CMD_READ_REQ/CNF –V1 Class 1 Read Request	0x2210/ 0x2211	133
6.1.10	PROFIBUS_FSPMM_CMD_WRITE_REQ/CNF – V1 Class 1 Write Request	0x2212/ 0x2213	136
6.1.11	PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND – Alarm Notification	0x221A	141
6.1.12	PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ/CNF – Alarm Acknowledge	0x221C/ 0x221D	144
6.1.13	PROFIBUS_FSPMM_CMD_DOWNLOAD_REQ/CNF – Download Slave Parameter Set	0x221E/ 0x221F	148
6.1.14	PROFIBUS_FSPMM_CMD_GLOBALCONTROL_REQ/CNF – Global Control Message	0x2220/ 0x2221	155
6.1.15	PROFIBUS_FSPMM_CMD_SLAVE_ACTIVATE_REQ/CNF – Activate/Deactivate Slaves	0x2226/ 0x2227	160
6.1.16	PROFIBUS_FSPMM_CMD_FAULT_IND – Indicate a Fatal Fault	0x2222	163

Table 53: Table 25: 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				
Area	Variable	Type	Value / Range	Description
Head	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 55: 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	x	Routing, do not change

Table 54: PROFIBUS\_FSPMM\_APP\_REG\_REQ – Application Register Request

#### Packet Status/Error

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

Table 55: 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				
Area	Variable	Type	Value / Range	Description
Head	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 57: 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	x	Routing, do not change

Table 56: PROFIBUS\_FSPMM\_APP\_REG\_CNF – Application Register Confirmation

**Packet Status/Error**

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

Table 57: 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.

### 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;
    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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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	x	Routing, do not change
Data	structure PROFIBUS_FSPMM_INIT_REQ_T			
	tBusParameter	STRUCT		Bus parameter settings

Table 58: 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x2201	PROFIBUS_FSPMM_CMD_INIT_CNF- Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change

Table 59: 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 5.2 “*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

```
#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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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
Data	structure PROFIBUS_FSPMM_SET_MODE_REQ_T			
	ulBusMode	UINT32	0,0x40,0x80,0xC0	Mode set that shall be set

Table 60: 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_MODE_CHANGE_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x2207	PROFIBUS_FSPMM_CMD_MODE_CHANGE_CNF_T - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	structure PROFIBUS_FSPMM_MODE_CHANGE_CNF_T			
	ulBusMode	UINT32		Active bus mode

Table 61: PROFIBUS\_FSPMM\_CMD\_SET\_MODE\_/CNF – 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 5.2 "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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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
Data	structure PROFIBUS_FSPMM_MODE_CHANGE_IND_T			
	ulBusMode	UINT32		Active bus mode

Table 62: 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  $\mu$ sec repeatedly to indicate that the slave is not present.

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

Explanation of Bits of <code>ulFlags</code> 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 <code>S1_Diag</code> will be cleared by execution of the get diag command
		1	Corresponding Diag Flag in <code>S1_Diag</code> will <b>not</b> be cleared by execution of the get diag command ( useful diagnostic tools )

Table 63: `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**

<b>structure PROFIBUS_FSPMM_PACKET_GET_SLAVE_DIAG_REQ_T</b>				
<b>Type: Request</b>				
<b>Area</b>	<b>Variable</b>	<b>Type</b>	<b>Value / Range</b>	<b>Description</b>
Head	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 7.1 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
	Data	structure PROFIBUS_FSPMM_GET_SLAVE_DIAG_REQ_T		
ulRemAdd		UINT32	0 ... 126	Requested slave address
ulFlag		UINT32	Bit mask	Flag register

Table 64: 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: Request				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x220B	PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	structure PROFIBUS_FSPMM_GET_SLAVE_DIAG_CNF_T			
	ulRemAdd	UINT32		Slave address
	tDiagnostic	PROFIBUS_FSPMM_DIAGNOSTIC_DATA_T		Slave diagnostic

Table 65: 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 100 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.



**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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 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	
Data	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 66: 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 should 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 5.3.1 "Cyclic Data Transfer" of this document.

#### 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				
Area	Variable	Type	Value / Range	Description
Head	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_FSPMM_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 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x220C	PROFIBUS_FSPMM_CMD_SET_OUTPUT_REQ_T - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	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 67: 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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 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	
Data	structure PROFIBUS_FSPMM_PACKET_SET_OUTPUT_CNF_T			
	ulRemAdd	UINT32	0...125	Address of slave

Table 68: 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_ADDRE SS (0xC038001DL)	Invalid slave address.

Table 69: 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 shall be used to read cyclic input data of a slave. Cyclic data transfer is discussed in more detail in section 5.3.1 "Cyclic Data Transfer" of this document.

#### 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				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ FSPMM_QUE	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 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
Data	structure PROFIBUS_FSPMM_GET_INPUT_REQ_T			
	ulRemAdd	UINT32	0...125	Slave Address from which to receive data

Table 70: 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)

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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x220F	PROFIBUS_FSPMM_CMD_GET_INPUT_CNF_T - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	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 71: 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 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 spend.

```
#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 the input data internaly 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.

### 6.1.9 PROFIBUS\_FSPMM\_CMD\_READ\_REQ/CNF –V1 Class 1 Read Request

This packet transfers a read request for a data block to a DP V1-Slave. It operates based on slot and index information.

For more information refer to section 5.3.1 *"Cyclic Data Transfer"*.

Explanations of the parameters:

1. The parameter `ulRemAdd = Rem_Add` contains the station address of the slave responder.
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.

## 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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	
Data	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	0 ...240	Requested data length

Table 72: PROFIBUS\_FSPMM\_CMD\_READ\_REQ –V1 Class 1 Read Request

## Packet Structure Reference

```
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 Description

structure PROFIBUS_FSPMM_PACKET_READ_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
Head	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
	ulId	UINT32	0 ... $2^{32}-1$	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x2211	PROFIBUS_FSPMM_CMD_READ_CNF_T - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	structure PROFIBUS_FSPMM_READ_CNF_T			
	ulRemAdd	UINT32	0 ...125	Slave address
	ulSlot	UINT32	0 ...254	Slot
	ulIndex	UINT32	0 ...254	Index
	abData	UINT8[]		Requested data

Table 73: PROFIBUS\_FSPMM\_CMD\_READ\_CNF –Confirmation of V1 Class 1 Read Request

### 6.1.10 PROFIBUS\_FSPMM\_CMD\_WRITE\_REQ/CNF – V1 Class 1 Write Request

This packet can be used to send a DP V1 Class 1 write request. It operates based on slot and index information.

Explanations of the parameters of the request packet:

- 1) The parameter `ulRemAdd = Rem_Add` contains the station address of the slave responder.
- 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 destination data block length is less than required then the reply will contain an error message. If the data block length is greater or equal than the required length, the reply contains the number of bytes, which have been written. The PROFIBUS DP-Slave may answer with an error response if the data access is not allowed.

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. For more information refer to section 5.3.1 "Cyclic Data Transfer".



**Note:** The three 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				
Area	Variable	Type	Value / Range	Description
Head	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 + PROFIBUS_FSPMM_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 7.1 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
Data	structure PROFIBUS_FSPMM_WRITE_REQ_T			
	ulRemAdd	UINT32	0 ... 125	Slave address
	ulSlot	UINT32	0 ... 254	Slot
	ulIndex	UINT32	0 ... 254	Index
	abData	UINT8[]		Write data

Table 74: PROFIBUS\_FSPMM\_CMD\_WRITE\_REQ– V1 Class 1 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;
}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_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: Request				
Area	Variable	Type	Value / Range	Description
Head	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 15	Packet data length in bytes Only in case of error TLR_E_PROFIBUS_FSPMM_MSAC1_NRS
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x2213	PROFIBUS_FSPMM_CMD_WRITE_CNF_T - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	structure PROFIBUS_FSPMM_WRITE_CNF_T			
	ulRemAdd	UINT32	0...125	Slave address
	ulSlot	UINT32	0...254	Slot
	ulIndex	UINT32	0...254	Index
	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 5.3.2.2. of this document
bErrorCode2	UINT8	0...255	ErrorCode2, meaning depends on bErrorCode1	

Table 75: PROFIBUS\_FSPMM\_CMD\_WRITE\_CNF – Confirmation of V1 Class 1 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 DPV1 support
	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 _FSPMM_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 76: 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. The `abDiagnostic_User_Data[ ]` field will contain the received diagnostic information from the DPV1-Slave transparently.
2. The `ulRem_Adr` parameter contains the station address, which indicates the alarm.
3. `ulAlarm_Type` identifies the alarm type. Coding see in the table below:

#### Alarm\_Type

Alarm_Type value	explanation
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 77: Available Alarm Types

4. `ulSlot_Number = Slot_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 norm description.
5. The `ulSpecifier = Alarm_Spec_Ack` parameter gives additional alarm information, e.g. an alarm appears, disappears or no further differentiation is not possible or if the alarm requires an additional user acknowledge. See the table below for further explanations.

**Alarm\_Spec\_Ack**

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

Table 78: Flags of Alarm\_Spec\_Ack parameter

The Alarm\_Specifier bits D1 and D0 contain the following information:

**Alarm\_Specifier bits D1 and D0**

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 79: Meaning of Alarm\_Specifier Bits D1 and D0

The Add\_Ack bit D2 contains the following information:

This alarm requires a separate user acknowledge additionally to the usual response. This can be done for instance by means of a write service.

**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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 Error Codes of the FSPMM-Task
	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	
Data	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 77: Available 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_Us er_Data	UINT8[]		Alarm diagnostic user data

Table 80: 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 message 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.

#### **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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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
Data	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 77: Available 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 81: PROFIBUS\_FSPMM\_CMD\_ALARM\_ACK\_REQ – Alarm Acknowledge

**Packet Structure Reference**

```

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				
Area	Variable	Type	Value / Range	Description
Head	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		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x221D	PROFIBUS_FSPMM_CMD_ALARM_ACK_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change	
Data	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 77: Available 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 82: PROFIBUS\_FSPMM\_CMD\_ALARM\_ACK\_CNF – Confirmation of Alarm Acknowledge

With this answer message, the host gets the confirmation for its previously sent PROFIBUS\_FSPMM\_CMD\_ALARM\_ACK\_REQ. If no error is indicated, the master has successfully sent back the PROFIBUS message to the slave device.

The following table may be useful for troubleshooting purposes:

ulSta	Description	Error source	Help
TLR_E_PROFIBUS_FSPMM_ALARM_NOT_INIT (0xC0380007L)	The alarm handler is not initialized	Device	No DPV1 capable device configured within the card
TLR_E_PROFIBUS_FSPMM_ALARM_NOT_STARTED (0xC0380008L)	The alarm handler is currently stopped	Device	No DPV1 capable slave device is in process data exchange with the Device. Check if network is running
TLR_E_PROFIBUS_FSPMM_ALARM_NOT_ENABLED (0xC0380009L)	The alarm that shall be acknowledged is not enabled in slave parameter data	Host	Enable the corresponding alarm in slave configuration data set
TLR_E_PROFIBUS_FSPMM_ALARM_NOT_PENDING (0xC038000AL)	The alarm that shall be Acknowledge is not pending on a PROFIBUS_FSPMM_CMD_ALARM_ACK_REQ	Host	Check the parameter Alarm_Type and Seq_Nr. Both must be equal to the PROFIBUS_FSPMM_CMD_ALARM_NOTIFICATION_IND parameter

Table 83: Possible Errors and Descriptions

### 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 4.2.4 “Detailed Description of Slave Parameters” of this document.

#### **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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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
	Data	structure PROFIBUS_FSPMM_DOWNLOAD_REQ		
ulRemoteAddresses		UINT32	0xFF	Local master
ulAreaCode		UINT32	0...125	Slave address
abData		UINT8[]		Slave parameter set, see structure below and detailed description in section 4.2.4.

Table 84: 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
bMaxChannelDataLen	UINT8	4...244	Maximum size of channel data
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 85: 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 decide 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 86: `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			
Message header	variable	type	value	signification
	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
	usParameterSetLength	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 87: 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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x221F	PROFIBUS_FSPMM_CMD_DOWNLOAD_CNF- Command
	ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change	
Data	structure PROFIBUS_FSPMM_DOWNLOAD_CNF			
	ulRemoteAddress	UINT32	0xFF	Address of master
	ulAreaCode	UINT32	0...125	Slave address

Table 88: 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 5.3.1.3 and 5.3.1.4 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

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

Table 89: 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 90: 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.

Bit D1 "Clear\_Data" can not 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/OPERATE). 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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
Data	structure PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T			
	ulRemAdd	UINT32	0—126,127	Slave address (127 represents the global address)
	ulSyncCommand	UINT32		Sync command
	ulFreezeCommand	UINT32		Freeze command
	ulGroupSelect	UINT32	0-7	Group selection

Table 91: 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 Error Codes of the FSPMM-Task
	ulCmd	UINT32	0x2221	PROFIBUS_FSPMM_CMD_GLOBALCONTROL_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change

Table 92: 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 whole 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.1 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
Data	structure PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T			
	ulRemAdd	UINT32	0—125	Slave address
	ulActivate	UINT32	1 2	Activate Slave Deactivate Slave

Table 93: 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: Request				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.1 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
Data	structure PROFIBUS_FSPMM_GLOBALCONTROL_REQ_T			
	ulRemAdd	UINT32	0—125	Slave address
	ulActivate	UINT32	1 2	Activate Slave Deactivate Slave

Table 94: 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 behaviour. 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				
Area	Variable	Type	Value / Range	Description
Head	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 <i>Packets</i> .
	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	
Data	PROFIBUS_FSPMM_FAULT_IND_T			
	ulRemAdd	UINT32	0 ... 127	Slave address where the fault occurred
	ulFault	UINT32		Number of fault

Table 95: PROFIBUS\_FSPMM\_CMD\_FAULT\_IND – Indicate a fatal Fault

## 6.2 The FSPMM2-Task

Within the DP-Master Stack, the FSPMM2-Task coordinates the underlying DP-Master state machines for DP V1 Class 2 Masters used for processing of the various services. This allows to maintain communication relationships between the Hilscher device acting as a Profibus DP V1 Class 2 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 96: 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_SENDFPACKET_FIFO/LIFO()` for sending a packet to the FSPMM-Task.



**Note:** DP V1 Class 2 services are not available in the current version of the Profibus DP Master protocol stack, but are already described in this manual as they will be supported by a release of the DP Master stack following soon.

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

<b>Overview over Packets of the FSPMM2 -Task</b>			
<b>No. of section</b>	<b>Packet</b>	<b>Command code (REQ/CNF or IND/RES)</b>	<b>Page</b>
6.2.1	PROFIBUS_FSPMM2_CMD_INIT_REQ/CNF – Initialization Command	0x4400/ 0x4401	167
6.2.2	PROFIBUS_FSPMM2_CMD_INITIATE_REQ/CNF– Initiate DPV1C2 Connection	0x4404/ 0x4405	170
6.2.3	PROFIBUS_FSPMM2_CMD_READ_REQ/CNF - V1 Class 2 Read Request	0x4406/ 0x4407	179
6.2.4	PROFIBUS_FSPMM2_CMD_WRITE_REQ/CNF - V1 Class 2 Write Request	0x4408/ 0x4409	184
6.2.5	PROFIBUS_FSPMM2_CMD_DATA_TRANSPORT_REQ/CNF – Combined V1 Class 2 Read and Write Request	0x440A/ 0x440B	191
6.2.6	PROFIBUS_FSPMM2_CMD_ABORT_REQ/CNF – Request Abort of Connection	0x440C/ 0x440D	198
6.2.7	PROFIBUS_FSPMM2_CMD_READ_SLAVE_DIAG_REQ/CNF – Read Slave Diagnostics (DP V1 Class 2)	0x440E/ 0x440F	203
6.2.8	PROFIBUS_FSPMM2_CMD_READ_INPUT_REQ/CNF – Read Input Values	0x4410/ 0x4411	206
6.2.9	PROFIBUS_FSPMM2_CMD_READ_OUTPUT_REQ/CNF – Read Output Values	0x4412/ 0x4413	209
6.2.10	PROFIBUS_FSPMM2_CMD_GET_CFG_REQ/CNF – Get Configuration Command	0x4414/ 0x4415	212
6.2.11	PROFIBUS_FSPMM2_CMD_SET_SLAVE_ADD_REQ/CNF – Set Slave Address (DP V1 Class 2)	0x4416/ 0x4417	215
6.2.12	PROFIBUS_FSPMM2_CMD_GET_MASTER_DIAG_REQ/CNF – Get Master Diagnosis	0x4418/ 0x4419	218
6.2.13	PROFIBUS_FSPMM2_CMD_LIVE_LIST_REQ/CNF – Live List Command	0x4430/ 0x4431	221
6.2.14	PROFIBUS_FSPMM2_CMD_ABORT_IND/RES – Abort Indication	0x4426/ 0x4427	224
6.2.15	PROFIBUS_FSPMM2_CMD_CLOSED_IND/RES – Closed Indication	0x4428/ 0x4429	228

*Table 97: Table 25: 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 of the stack. For this purpose, a complete bus parameter set needs to be transferred. For a detailed description of all members of the bus parameters settings structure `tBusParameter` see section 4.2.3 “Detailed Description of Bus and Master Parameters”.

The corresponding confirmation packet does not have any parameters.



**Note:** Setting of bus parameters is not possible in state OFFLINE.

### Packet Structure Reference

```
typedef struct PROFIBUS_DL_BUS_PARAMETER_SET_Ttag {
    TLR_UINT16 usBus_Para_Len; /* 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  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;

typedef struct PROFIBUS_FSPMM2_INIT_REQ_Ttag
{
    PROFIBUS_DL_BUS_PARAMETER_SET_T tBusParameter;
} PROFIBUS_FSPMM2_INIT_REQ_T;

typedef struct PROFIBUS_FSPMM2_PACKET_INIT_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;
    PROFIBUS_FSPMM2_INIT_REQ_T tData;
} PROFIBUS_FSPMM2_PACKET_INIT_REQ_T;

#define PROFIBUS_FSPMM2_INIT_REQ_SIZE (sizeof(PROFIBUS_FSPMM2_INIT_REQ_T) -
sizeof(PROFIBUS_DL_BUS_PARAMETER_SET_T))
```

**Packet Description**

structure PROFIBUS_FSPMM2_PACKET_INIT_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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	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 7.2 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
tData	structure PROFIBUS_FSPMM2_INIT_REQ_T			
	tBusParameter	PROFIBUS_DL_BUS_PARAMETER_SET_T		Bus Parameter Data as described in section 0

Table 98: PROFIBUS\_FSPMM2\_CMD\_INIT\_REQ- Initialization Command for Class 2 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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 99: PROFIBUS\_FSPMM2\_CMD\_INIT\_CNF - Confirmation of Initialization Command

## 6.2.2 PROFIBUS\_FSPMM2\_CMD\_INITIATE\_REQ/CNF– Initiate DPV1C2 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 - V1 Class 2 Read Request
- PROFIBUS\_FSPMM2\_CMD\_WRITE\_REQ/CNF - V1 Class 2 Write Request
- PROFIBUS\_FSPMM2\_CMD\_DATA\_TRANSPORT\_REQ/CNF – Combined V1 Class 2 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 class 2 connection (MSAC\_C2 connection) which the DP V1 Master Class 2 requests from the slave. The supervision based on this period is performed as long as the DPV1 class 2 connection is active. The time out values are specified in units of 10 milliseconds.

According to the Profibus specification, Profibus DP V1 Master Class 2 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, parallelly established connections 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. All those informations originate from the slave itself.
- In case of failure the confirmation packet will be structured as described under `PROFIBUS_FSPMM2_INITIATE_CNF_NEG_T`. For an explanation of the error handling see section 5.3.2.2 "**DP V1 Error Processing**" of this document and the packet description.

For more information also see section 5.4 "*Functionality of the FSPMM2-Task*" of this manual.

**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				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	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
	bFeaturesSupported1	UINT8	0...255	Bit field containing information about the service functionality
	bFeaturesSupported2	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
	usProfileIdentifier	UINT16	0...65535	Identification number for profile (A value of 0 indicates no profile is currently used.)
tAddAddrParam	PROFIBUS_FSPMM2_ADDR_PARAM		Additional Address Parameter Table	

Table 100: 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_POS_T				
Type: Positive Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	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
	usProfileIdentificationNumber	UINT16	0...65535	Identification number for the actually chosen profile
	tAddAddrParam	PROFIBUS_FSPMM2_ADD_ADDR_PARAMETER		Additional Address Parameter Table

Table 101: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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	
tData	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 5.3.2.2.
	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 102: PROFIBUS\_FSPMM2\_CMD\_INITIATE\_CNF\_NEG - Negative Confirmation of Initiate Command

### 6.2.3 PROFIBUS\_FSPMM2\_CMD\_READ\_REQ/CNF - V1 Class 2 Read Request

This packet transfers a read request for a data block from the device acting as a Profibus DP V1 Master Class 2 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 5.3.2.2 "**DP V1 Error Processing**" of this document 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure PROFIBUS_FSPMM2_READ_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 read.

Table 103: 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_POS_T				
Type: Positive Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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	
tData	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 104: PROFIBUS\_FSPMM2\_CMD\_READ\_CNF\_POS - Positive Confirmation of Read Command

**Packet Status/Error**

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

Table 105: 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_NEG_T				
Type: Negative Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	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 5.3.2.2.
	bErrorCode2	UINT8	0 ... 255	ErrorCode2, meaning depends on bErrorCode1
bReserved	UINT8			

Table 106: PROFIBUS\_FSPMM2\_CMD\_READ\_CNF\_NEG - Negative Confirmation of Read Command

## 6.2.4 PROFIBUS\_FSPMM2\_CMD\_WRITE\_REQ/CNF - V1 Class 2 Write Request

This packet can be used to send a DP V1 Class 2 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 5.3.2.2 "**DP V1 Error Processing**" of this document 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_CMD_WRITE_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	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[]		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 107: 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_POS_T				
Type: Positive Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	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 108: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	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 5.3.2.2.
	bErrorCode2	UINT8	0 ... 255	ErrorCode2, meaning depends on value of bErrorCode1
bReserved	UINT8		Reserved	

Table 109: PROFIBUS\_FSPMM2\_CMD\_WRITE\_CNF\_NEG - Negative Confirmation of Write Command

## 6.2.5 PROFIBUS\_FSPMM2\_CMD\_DATA\_TRANSPORT\_REQ/CNF – Combined V1 Class 2 Read and Write Request

This packet can be used to send a data transport request, i.e. combined DP V1 Class 2 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 5.3.2.2 “**DP V1 Error Processing**” of this document 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	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 110: 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				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ FSPMM2_QUE	Destination queue handle, unchanged
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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	
tData	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 111: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	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 5.3.2.2.
	bErrorCode2	UINT8	0 ... 255	ErrorCode2, meaning depends on bErrorCode1
	bReserved	UINT8		

Table 112: 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 class 2 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 113: 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 114: 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
			1	1		DDL	ABT_SE
			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_ST O	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 115: 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_CMD_ABORT_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	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 116: 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_CMD_ABORT_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 Error Codes of the FSPMM2-Task
	ulCmd	UINT32	0x440D	PROFIBUS_FSPMM2_CMD_ABORT_RCNF - Command
	ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change	
tData	structure			
	ulCRef	UINT32		Connection Reference

Table 117: 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 Class 2)

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 5.3.5.2 “*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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	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 118: PROFIBUS\_FSPMM2\_CMD\_READ\_SLAVE\_DIAG\_REQ - Read Slave Diagnostics (DP V1 Class 2)

**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				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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	
tData	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 119: PROFIBUS\_FSPMM2\_CMD\_READ\_SLAVE\_DIAG\_CNF - Confirmation of Read Slave Diagnostics (DP V1 Class 2)

## 6.2.8 PROFIBUS\_FSPMM2\_CMD\_READ\_INPUT\_REQ/CNF – Read Input Values

This packet provides the Class 2 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure PROFIBUS_FSPMM2_READ_INPUT_REQ_T			
	ulRemAdd	UINT32	0...126	Remote Address of the slave to read input data from

Table 120: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	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 121: 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 Class 2 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_CMD_READ_OUTPUT_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	ulRemAdd	UINT32	0...126	Remote address of Profibus DP slave station from which output data shall be requested.

Table 122: 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_CMD_READ_OUTPUT_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	structure			
	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 123: 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 5.3.3 "**Configuration of Inputs and Outputs**" of this document. 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_CMD_GET_CFG_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	ulRemAdd	UINT32	0...126	Remote address of Profibus DP slave station from which configuration information shall be requested.

Table 124: 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_CMD_GET_CFG_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	structure			
	ulRemAdd	UINT32	0...126	Remote address of Profibus DP slave station from which configuration information shall be requested.
	abData	UINT8[]		Data area containing configuration information

Table 125: 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 Class 2)

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_CMD_SET_SLAVE_ADD_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	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 126: 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_CMD_SET_SLAVE_ADD_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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	
tData	structure			
	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 127: 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 class 1.

### *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_CMD_GET_MASTER_DIAG_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	ulRemAdd	UINT32	0...125	Remote address of Profibus DP Master station from which diagnostic data shall be requested.
	blIdentifier	UINT8	0...255	Identifier
	abReserved[3]	UINT8 []		Reserved

Table 128: 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_CMD_GET_MASTER_DIAG_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	structure			
	ulRemAdd	UINT32	0...125	Remote address
	bIdentifier	UINT8	0...255	Identifier
	abReserved[3]	UINT8 []		Reserved
	abData	UINT8 []		Diagnostic data

Table 129: 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 130: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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 131: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	structure PROFIBUS_FSPMM2_LIVE_LIST_CNF_T			
	abLiveList	UINT8[]		Live list data

Table 132: 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 Class 2. 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 113: 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 114: 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 115: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	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 133: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	ulCRef	UINT32		Connection reference

Table 134: PROFIBUS\_FSPMM2\_CMD\_ABORT\_RES – Response to Abort Indication

### 6.2.15 PROFIBUS\_FSPMM2\_CMD\_CLOSED\_IND/RES – Closed Indication

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_CMD_CLOSED_IND				
Type: Indication				
Area	Variable	Type	Value / Range	Description
tHead	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 7.2 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
tData	structure			
	ulCRef	UINT32		Connection reference of the closed connection

Table 135: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.2 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
tData	structure PROFIBUS_FSPMM2_CLOSED_RES_T			
	ulCRef	UINT32		Connection reference

Table 136: PROFIBUS\_FSPMM2\_CMD\_CLOSED\_RES –Response to Closed Indication

## 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 137: 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_SENDFPACKET_FIFO/LIFO()` for sending a packet to the DL-Task.

In detail, the following functionality is provided by the `PROFIBUS DL-Task`:

<b>Overview over Packets of the PROFIBUS DL -Task</b>			
<b>No. of section</b>	<b>Packet</b>	<b>Command code (REQ/CNF or IND/RES)</b>	<b>Page</b>
6.3.1	PROFIBUS_DL_CMD_END_PROCESS_REQ/CNF– Finish the DL Task	0x100/ 0x101	233
6.3.2	PROFIBUS_DL_CMD_START_DLE_REQ/CNF – Starts the DL-Layer	0x102/ 0x103	235
6.3.3	PROFIBUS_DL_CMD_STOP_DLE_REQ/CNF – Stop DL Layer	0x104/ 0x105	237
6.3.4	PROFIBUS_DL_CMD_DATA_ACK_REQ/CNF - Send SDA Service	0x106/ 0x107	239
6.3.5	PROFIBUS_DL_CMD_DATA_ACK_IND - Receive SDA Service	0x108	245
6.3.6	PROFIBUS_DL_CMD_DATA_REQ/CNF - Send SDN Service	0x10A/ 0x10B	248
6.3.7	PROFIBUS_DL_CMD_DATA_IND - Receive SDN Service Indication	0x10C	254
6.3.8	PROFIBUS_DL_CMD_DATA_REPLY_REQ/CNF - Send SRD Service	0x10E/ 0x10F	257
6.3.9	PROFIBUS_DL_CMD_DATA_REPLY_IND - Receive SRD Service Indication	0x110	264
6.3.10	PROFIBUS_DL_CMD_DATA_REPLY_UPDATE_REQ/CNF - Reply Update Service	0x112/ 0x113	267
6.3.11	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_REQ/CNF - Activate an SAP for Request	0x120/ 0x121	271
6.3.12	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_RESPONDER_REQ/CNF - Activate an SAP for Responder Functionality	0x122/ 0x123	276
6.3.13	PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_REQ/CNF - Deactivate an SAP for Request	0x128/ 0x129	281
6.3.14	PROFIBUS_DL_CMD_SET_VALUE_BUS_PARAMETER_SET_REQ/CNF - Load the Bus Parameter Set	0x126/ 0x127	284
6.3.15	PROFIBUS_DL_CMD_SET_VALUE_REQ/CNF - Set Value Service	0x12A/ 0x12B	290
6.3.16	PROFIBUS_DL_CMD_SEND_FDL_STATUS_REQ/CNF – Obtain FDL Status	0x132/ 0x133	293

Table 138: Overview over the Packets of the PROFIBUS DL -Task of the PROFIBUS DP-Master Protocol Stack

### 6.3.1 PROFIBUS\_DL\_CMD\_END\_PROCESS\_REQ/CNF– Finish the DL Task

This packet shall be sent to close the DL-Task. All resources will be deallocated and the task finishes.



**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_END_PROCESS_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_CMD_END_PROCESS_REQ_T;
```

#### Packet Description

Type: Request				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	PB_DL_QUE	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.3 Error Codes of the DL-Task
	ulCmd	UINT32	0x100	PROFIBUS_DL_CMD_END_PROCESS_REQ - Command
	ulExt	UINT32	0	Extension (not in use, set to zero for compatibility reasons)
	ulRout	UINT32	x	Routing, do not change

Table 139: PROFIBUS\_DL\_CMD\_END\_PROCESS\_REQ– Finish the DL Task

### Packet Structure Reference

```
typedef struct PROFIBUS_DL_PACKET_CMD_END_PROCESS_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;
} PROFIBUS_DL_PACKET_CMD_END_PROCESS_CNF_T;
```

### Packet Description

Type: Confirmation				
Area	Variable	Type	Value / Range	Description
Head	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 section 7.3 Error Codes of the DL-Task
	ulCmd	UINT32	0x101	PROFIBUS_DL_CMD_END_PROCESS_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
ulRout	UINT32	x	Routing, do not change	

Table 140: PROFIBUS\_DL\_CMD\_END\_PROCESS\_CNF– Finish the DL Task

### 6.3.2 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

Type: Request				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	PB_DL_QUE	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.3 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 141: 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

Type: Request				
Area	Variable	Type	Value / Range	Description
Head	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	x	Routing, do not change	

Table 142: PROFIBUS\_DL\_CMD\_START\_DLE\_CNF – Confirmation of Start the DL-Layer Request

### 6.3.3 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

Type: Request				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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 143: 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**

Type: Confirmation				
Area	Variable	Type	Value / Range	Description
Head	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	x	Routing, do not change	

Table 144: PROFIBUS\_DL\_CMD\_STOP\_DLE\_CNF – Confirmation of Stop DL Layer Request

### 6.3.4 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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
Data	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  <b>Note:</b> 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 145: 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: Request				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -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	x	Routing, do not change
Data	structure PROFIBUS_DL_DATA_ACK_CNF_T			
	bSrvCls	UINT8	0,1	Service Class
	bDstAddr	UINT8	0-126	Destination address <b>Note:</b> 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 146: 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. <u>Action:</u> 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 147: 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.  <u>Action:</u> 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.  <u>Action:</u> 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.  <u>Action:</u> Check slave for correct operation.
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	Slave	No response of the station at all  <u>Action:</u> 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.  <u>Action:</u> Check master DP-Address or highest-station-address of other masters Examine bus wiring to avoid bus short circuits.

Table 148: Reported Errors, their Sources and Possible Actions

### 6.3.5 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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
Data	structure PROFIBUS_DL_DATA_ACK_IND_T			
	bSrvCls	UINT8	0,1	Service Class
	bDstAddr	UINT8	0-126	Destination address <b>Note:</b> 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 <b>Note:</b> 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 149: 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`

### 6.3.6 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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
Data	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 <b>Note:</b> 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 150: 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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See Table 152: PROFIBUS_DL_CMD_DATA_CNF - Packet Status/Error
	ulCmd	UINT32	0x10B	PROFIBUS_DL_CMD_DATA_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	structure PROFIBUS_DL_DATA_CNF_T			
	bSrcCls	UINT8	0,1	Service Class
	bDstAddr	UINT8	0-126, 127 (= Broadcast)	Destination address <b>Note:</b> 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 151: 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 152: 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.  <u>Action:</u> 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. <u>Action:</u> Check master DP-Address or highest-station-address of other masters Examine bus wiring to avoid bus short circuits.

*Table 153: Reported Errors, their Sources and Possible Actions*

### 6.3.7 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: Request				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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	
Data	structure PROFIBUS_DL_DATA_IND_T			
	bSrcCls	UINT8	0,1	Service Class
	bDstAddr	UINT8	0-126, 127 (= Broadcast)	Destination address <b>Note:</b> 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
	bSrcAddr	UINT8	0-126	Source address <b>Note:</b> 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 154: 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`

### 6.3.8 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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
Data	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 <b>Note:</b> 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 155: 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				
Area	Variable	Type	Value / Range	Description
Head	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 157: 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	x	Routing, do not change	
Data	structure PROFIBUS_DL_DATA_REPLY_CNF_T			
	bSrvCls	UINT8	0,1	Service Class
	bDstAddr	UINT8	0-126	Destination address <b>Note:</b> 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 156: 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 157: 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.  <u>Action:</u> 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.  <u>Action:</u> 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.  <u>Action:</u> 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, as the slave did not send back any data.  <u>Action:</u> Check slave for correct operation.
TLR_E_PROFIBUS_DL_ACK_NA (0xC0060088L)	Slave	No response of the station at all  <u>Action:</u> 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. <u>Action:</u> Check master DP-Address or highest-station-address of other masters Examine bus wiring to avoid bus short circuits.

Table 158: Reported Errors, their Sources and Possible Actions

### 6.3.9 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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.3 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	
Data	structure PROFIBUS_DL_DATA_REPLY_IND_T			
	bSrvCls	UINT8	0,1	Service Class
	bDstAddr	UINT8	0-126	Destination Address <b>Note:</b> 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 <b>Note:</b> 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 159: 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
<code>PROFIBUS_DL_UPDATE_STATUS_NO</code>	0x00	No data transmitted in response
<code>PROFIBUS_DL_UPDATE_STATUS_LO</code>	0x01	Low priority data transmitted in response
<code>PROFIBUS_DL_UPDATE_STATUS_HI</code>	0x02	High priority data transmitted in response

Table 160: Allowed Values and their Meanings for Variable `bUpdateStatus`

### 6.3.10 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_DATA_REPLY_UPDATE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
Head	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	
Data	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 161: 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_DATA_REPLY_UPDATE_CNF_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -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	x	Routing, do not change	
Data	structure PROFIBUS_DL_DATA_REPLY_UPDATE_CNF_T			
	bSrcSAPIdx	UINT8	0-62, 64 = NIL SAP	Source Service Access Point
	bSrvCls	UINT8	0,1	Service Class

Table 162: PROFIBUS\_DL\_CMD\_DATA\_REPLY\_UPDATE\_CNF – Confirmation of Reply Update Service

### 6.3.11 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 5.5.2 “*Management of Services Access Points*” on page 102 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 6.3.4, 6.3.6 or 6.3.8. If later on this SAP is not needed any longer, it may be deactivated using the packet “SAP Deactivate” described in section 5.5.2.4 „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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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
Data	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 163: 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 164: 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 165: 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								
<code>bMaxDLSDULenReqLow</code>	x	0	x	0	0	0	x	0	x	0	x	0	x	x	x
<code>bMaxDLSDULenReqHigh</code>	0	x	x	0	0	0	0	x	0	x	0	x	x	x	x
<code>bMaxDLSDULenIndCnfLow</code>	0	0	0	x	0	x	x	0	0	x	x	x	x	0	x
<code>bMaxDLSDULenIndCnfHigh</code>	0	0	0	0	x	x	0	x	x	0	x	x	0	x	x

*Table 166: 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												
<code>bMaxDLSDULenReqLow</code>	0	0	0	x	0	0	x	x	x	x	0	x	
<code>bMaxDLSDULenReqHigh</code>	0	0	0	0	x	x	0	x	x	0	x	x	
<code>bMaxDLSDULenIndCnfLow</code>	x	0	x	x	0	x	0	x	0	x	x	x	
<code>bMaxDLSDULenIndCnfHigh</code>	0	x	x	0	x	0	x	0	x	x	x	x	

*Table 167: 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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.3 Error Codes of the DL-Task.
	ulCmd	UINT32	0x121	PROFIBUS_DL_CMD_DLSAP_ACTIVATE_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	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 168: PROFIBUS\_DL\_CMD\_DLSAP\_ACTIVATE\_CNF - Confirmation of Activate a SAP for Request

### 6.3.12 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 5.5.2 “*Management of Services Access Points*”, 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 5.5.2 “*Management of Services Access Points*” of this manual.

#### 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 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
Data	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 169: 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											
<code>bMaxDLSDULenReqLow</code>	x	0	x	x	0	0	x	x	x	x	0	x
<code>bMaxDLSDULenReqHigh</code>	0	x	x	0	x	x	0	x	x	0	x	x
<code>bMaxDLSDULenIndLow</code>	0	0	0	x	0	x	0	x	0	x	x	x
<code>bMaxDLSDULenIndHigh</code>	0	0	0	0	x	0	x	0	x	x	x	x

*Table 170: Allowed combinations of `bMaxDLSDULenReqLow`, `bMaxDLSDULenReqHigh`, `bMaxDLSDULenIndLow` and `bMaxDLSDULenIndHigh` 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 <code>bAcc</code> parameter. It is used in the following manner: <ul style="list-style-type: none"> <li>■ The <code>bAcc</code> parameter will be changed according to its setting within the packet.</li> <li>■ All other parameters will completely remain unchanged.</li> </ul>

Table 171: 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				
Area	Variable	Type	Value / Range	Description
Head	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 <sup>32</sup> -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	x	Routing, do not change	
Data	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 172: PROFIBUS\_DL\_CMD\_DLSAP\_ACTIVATE\_RESPONDER\_CNF – Confirmation of Activate a SAP for Responder Functionality

### 6.3.13 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 described in section 6.3.11 or with the
- “PROFIBUS\_DL\_CMD\_DLSAP\_DEACTIVATE\_REQ/CNF - Deactivate an SAP for Request” packet described in section 6.3.12.

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**

<b>structure PROFIBUS_DL_PACKET_DLSAP_DEACTIVATE_REQ_T;</b>				
<b>Type: Request</b>				
<b>Area</b>	<b>Variable</b>	<b>Type</b>	<b>Value / Range</b>	<b>Description</b>
Head	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		<i>See section 7.3 Error Codes of the DL-Task.</i>
	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
Data	structure PROFIBUS_DL_DLSAP_DEACTIVATE_REQ_T			
	bSrcSAPIdx	UINT8	0-63, 64	Source Service Access Point to be deactivated

Table 173: 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				
Area	Variable	Type	Value / Range	Description
Head	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 7.3 Error Codes of the DL-Task.
	ulCmd	UINT32	0x129	PROFIBUS_DL_CMD_DLSAP_DEACTIVATE_CNF - Command
	ulExt	UINT32	0	Extension, unchanged
	ulRout	UINT32	x	Routing, do not change
Data	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 174: PROFIBUS\_DL\_CMD\_DLSAP\_DEACTIVATE\_CNF – Confirmation of Deactivate an SAP for Request

### 6.3.14 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 0 “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.

#### 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 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 ultTCL; /* 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				
Area	Variable	Type	Value / Range	Description
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	PB_DL_QUE	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet identification as unique number generated by the Source process of the packet
	ulSta	UINT32		See section 7.3 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	x	Routing, do not change
Data	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)  <b>Note:</b> 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
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 175: 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 ultTCL; /* Isochronous cycle time */
    TLR_UINT8 bMax_TSH; /* Maximum Shift Time */
} PROFIBUS_DL_BUS_PARAMETER_SET_T;

```

**Packet Description**

<b>structure PROFIBUS_DL_PACKET_SET_VALUE_BUS_PARAMETER_SET_CNF_T</b>				
<b>Type: Confirmation</b>				
<b>Area</b>	<b>Variable</b>	<b>Type</b>	<b>Value / Range</b>	<b>Description</b>
Head	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	
Data	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_Glob	UINT8		Maximum allowed parallel active USER Global Control

aI_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
uITCL	UINT32		Isochronous cycle time
bMax_TSH	UINT8		Maximum Shift Time

Table 176: PROFIBUS\_DL\_CMD\_SET\_VALUE\_BUS\_PARAMETER\_SET\_CNF – Confirmation of Loading the Bus Parameter Set

### 6.3.15 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<sub>SDR</sub>

Indicates the smallest station delay response time. See section 4.2.3.5 “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.



**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				
Area	Variable	Type	Value / Range	Description
Head	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	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 7.3 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
	Data	structure PROFIBUS_DL_SET_VALUE_REQ_T		
eVlu		ENUM	0,1	Value to be set: 0: MINTDSR 1: DLADDR

Table 177: 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				
Area	Variable	Type	Value / Range	Description
Head	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	x	Routing, do not change

Table 178: PROFIBUS\_DL\_CMD\_SET\_VALUE\_CNF – Confirmation of Set Value Service

### 6.3.16 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				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ PB_DL_QUE	Destination Queue-Handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.3 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
tData	structure PROFIBUS_DL_SEND_FDL_STATUS_REQ_T			
	bDstAddr	UINT8		Slave address whose status is to be requested

Table 179: 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				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section 7.3 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
tData	structure PROFIBUS_DL_SEND_FDL_STATUS_CNF_T			
	bStatus	UINT8		Requested status of addressed slave

Table 180: PROFIBUS\_DL\_CMD\_SEND\_FDL\_STATUS\_CNF – Confirmation of Obtain FDL Status 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 181: FSPMM-Task process queue

The diagnostic and error messages which can be issued by the APM-Task are described in section 7.4 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	296
6.4.2	PROFIBUS_APM_CMD_REDUNDANT_MODE_IND – Indicate Status changed of active Master	0x3002	299

Table 182: 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. For more information on this topic, please refer to section “*Redundancy Functionality*” on page 104 of this document.

#### **Packet Structure Reference**

```
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_CMD_REDUNDANT_MODE_REQ				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	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		
	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
tData	structure			
	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 1-7: reserved	Bit mask for redundant options

Table 183: 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_CMD_REDUNDANT_MODE_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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 7.4 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 184: 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 active master has changed at the local LAS. For more information on this topic, please refer to section “Redundancy Functionality” on page 104 of this document.

### Packet Structure Reference

```
typedef struct PROFIBUS_APM_REDUNDANT_MODE_IND_Ttag
{
    TLR_BOOLEAN8 fActiveMasterExist;
} 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_CMD_REDUNDANT_MODE_IND				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ APM_QUE	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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	1	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 <sup>32</sup> -1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32	0	See section 7.4 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	
tData	structure			
	fActiveMasterExist	UINT8	TRUE FALSE	Status of active master

Table 185: 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	301
6.5.2	RCX_ACTIVATE_DATABASE_REQ/CNF – Activate the new Configuration Database	0x2F84/ 0x2F85	304

Table 186: 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				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ rcX queue handle	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -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 187: 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_VERIFY_DATABASE_CNF_PACKET_SIZE sizeof(RCX_VERIFY_DATABASE_CNF_T)
```

**Packet Description**

structure RCX_VERIFY_DATABASE_CNF				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	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
tData	structure			
	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 can not 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 188: 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				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ rcX queue handle	Destination queue handle
	ulSrc	UINT32	0 ... 2 <sup>32</sup> -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 <sup>32</sup> -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 189: 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: Request				
Area	Variable	Type	Value / Range	Description
tHead	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 <sup>32</sup> -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
tData	structure			
	abSlvSt[16]	UINT8		Bit field which contains the addresses of slaves which are configured

Table 190: RCX\_ACTIVATE\_DATABASE\_CNF - Confirmation of Activate the new Configuration Database

## 7 Status/Error codes overview

### 7.1 Error Codes of the FSPMM-Task

Hexadecimal Value	Definition Description
0x0000000L	TLR_S_OK Status ok
0xC000001L	TLR_E_FAIL Common error, detailed error information optionally present in the data area of packet
0xC000007L	TLR_E_INVALID_PACKET_LEN Invalid packet length.
0xC00001AL	TLR_E_REQUEST_RUNNING A new request was received, but one request is still running.
0xC000101L	TLR_E_DATABASE_ACCESS_FAILED Database couldn't be opened. No verification possible.
0xC000102L	TLR_E_CIR_MASTER_PARAMETER_FAILED Parameters of the master were changed, which is not allowed.
0xC000103L	TLR_E_CIR_SLAVE_PARAMETER_FAILED Result of the verification: One or more slave parameter set have an error.
0xC038002L	TLR_E_PROFIBUS_FSPMM_INV_BUSMODE Invalid bus mode for this command.
0xC038003L	TLR_E_PROFIBUS_FSPMM_RESET FSPMM task is reset.
0xC038004L	TLR_E_PROFIBUS_FSPMM_ACLR Profibus Master is at auto clear state.
0xC038005L	TLR_E_PROFIBUS_FSPMM_CONTROL_TIMER_EXPIRED Data Control Timer expired. No bus access for sending global control.
0xC038006L	TLR_E_PROFIBUS_FSPMM_ALARM_OVERFLOW Alarm buffer overflow.
0xC038007L	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_INIT Alarm handler is not initialized.
0xC038008L	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_STARTED Alarm handler is not started.
0xC038009L	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_ENABLED Alarms are disabled.
0xC03800AL	TLR_E_PROFIBUS_FSPMM_ALARM_NOT_PENDING Alarm is not at a pending state.
0xC03800BL	TLR_E_PROFIBUS_FSPMM_ALARM_STATE_ERROR Invalid Alarm state.
0xC03800CL	TLR_E_PROFIBUS_FSPMM_ALARM_SEQ_ERROR Alarm sequence error occurred.
0xC03800DL	TLR_E_PROFIBUS_FSPMM_MSAC1_STATE_ERROR Alarm handler is not at the proper state.
0xC03800EL	TLR_E_PROFIBUS_FSPMM_MSAC1_FAULT Get an alarm acknowledge without an alarm.
0xC03800FL	TLR_E_PROFIBUS_FSPMM_INVALID_AREA_CODE

Hexadecimal Value	Definition Description
	Invalid area code or slave address received.
0xC0380012L	TLR_E_PROFIBUS_FSPMM_IV_BUS_PRM Invalid bus parameter received.
0xC0380013L	TLR_E_PROFIBUS_FSPMM_IV_SLAVE_PRM Invalid slave parameter received.
0xC0380014L	TLR_E_PROFIBUS_FSPMM_ACK_NO Command cannot be executed at the actual bus state.
0xC0380015L	TLR_E_PROFIBUS_FSPMM_ACK_GE Error while sending global control.
0xC0380016L	TLR_E_PROFIBUS_FSPMM_MSAL1_FAULT Failure at alarm handler. Alarm handler is stopped.
0xC0380017L	TLR_E_PROFIBUS_FSPMM_MSAC2_FAULT Failure at MSAC2 handler.
0xC0380018L	TLR_E_PROFIBUS_FSPMM_REJ_SE Device is stopping the communication or not in OPEN state.
0xC0380019L	TLR_E_PROFIBUS_FSPMM_REJ_PS A previous service is still in process.
0xC038001AL	TLR_E_PROFIBUS_FSPMM_REJ_LE Message has an invalid length.
0xC038001BL	TLR_E_PROFIBUS_FSPMM_REJ_IV Invalid parameter at request.
0xC038001CL	TLR_E_PROFIBUS_FSPMM_REJ_ABORT Device aborts DP V1 communication.
0xC038001DL	TLR_E_PROFIBUS_FSPMM_INVALID_SLAVE_ADDRESS Invalid slave address.
0xC038001EL	TLR_E_PROFIBUS_FSPMM_ALREADY_INITIALIZED FSPMM Already initialized.
0xC038001FL	TLR_E_PROFIBUS_FSPMM_INVALID_APPLICATION Command from not registered application.
0xC0380020L	TLR_E_PROFIBUS_FSPMM_DMPMM_IV_STATE Command not allowed in current state.
0xC0380021L	TLR_E_PROFIBUS_FSPMM_PB_FLAG_ERROR_ACTION_FLAG Function 'AUTO CLEAR' not supported.
0xC0380022L	TLR_E_PROFIBUS_FSPMM_PB_FLAG_ISO_MODE_MSK Function 'ISO_MODE' not supported.
0xC0380023L	TLR_E_PROFIBUS_FSPMM_DL_PB_FLAG_ISOM_SYNC Function 'ISO_MODE_SYNC' not supported.
0xC0380024L	TLR_E_PROFIBUS_FSPMM_DL_PB_FLAG_ISOM_FREEZE Function 'ISO_MODE_FREEZE' not supported.
0xC0380025L	TLR_E_PROFIBUS_FSPMM_MSAC1_NRS Negative response received.

Table 191: Error Messages of the FSPMM-Task

### 7.1.1 Diagnostic Codes of the FSPMM-Task

Hexadecimal Value	Definition Description
0xC0380001L	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_NO_INIT_PACKET Getting a packet from the pool to initialize the Data Link failed.
0xC0380002L	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_BUS_PARAMETER_REJECTED Command for setting the bus parameter is rejected.
0xC0380003L	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_SSAP_ACTIVATE_FAIL Activating the source SAP failed.
0xC0380004L	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_STATE_ERROR DMPMM state machine is at an invalid state.
0xC0380005L	TLR_DIAG_E_PROFIBUS_FSPMM_FSPMM_STATE_ERROR FSPMM state machine is at an invalid state.
0xC0380006L	TLR_DIAG_E_PROFIBUS_FSPMM_MSCY1M_STATE_ERROR MSCY1M state machine is at an invalid state.
0xC0380007L	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_SSAP_ACTIVATE_RESPONDER_FAIL Activate RSAP failed.
0xC0380008L	TLR_DIAG_E_PROFIBUS_FSPMM_DMPMM_NO_GLBCNTRL_PACKET Getting a packet from the pool to send global control failed.
0xC0380009L	TLR_DIAG_E_PROFIBUS_FSPMM_NOT_SUPPORTED Function is not supported at the current version of the stack.
0xC038000AL	TLR_DIAG_E_PROFIBUS_FSPMM_NO_DATABASE No database found.
0xC038000BL	TLR_DIAG_E_PROFIBUS_FSPMM_INVALID_DATABASE Getting a packet from the pool to initialize the Master state machine failed.
0xC038000CL	TLR_DIAG_E_PROFIBUS_FSPMM_UNKNOWN_TELEGRAM Telegram with an unknown SAP was received.
0xC038000EL	TLR_DIAG_E_PROFIBUS_FSPMM_UNKNOWN_DPV1_CMD Answer of an invalid DP V1 service received.
0xC0380010L	TLR_DIAG_E_PROFIBUS_FSPMM_TASK_PRM_VERSION_INVALID Invalid version of Task Parameters.

Table 192: Diagnostic Messages of the FSPMM-Task

## 7.2 Error Codes of the FSPMM2-Task

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
0xC0690001L	TLR_E_PROFIBUS_FSPMM2_LENGTH_INVALID Invalid data length.
0xC0690002L	TLR_E_PROFIBUS_FSPMM2_NOT_IMPLEMENTED Service not implemented.
0xC0690003L	TLR_E_PROFIBUS_FSPMM2_ADD_INVALID Invalid address.
0xC0690004L	TLR_E_PROFIBUS_FSPMM2_SERVICE_IN_REQUEST Previous service already in request.
0xC0690005L	TLR_E_PROFIBUS_FSPMM2_NOT_IN_OPEN_STATE Connection is not in state open.
0xC0690006L	TLR_E_PROFIBUS_FSPMM2_OUT_OF_RESOURCES Out of resources for new connections.
0xC0690007L	TLR_E_PROFIBUS_FSPMM2_IN_USE Connection to this slave already in use.
0xC0690008L	TLR_E_PROFIBUS_FSPMM2_ALREADY_INIT Stack is already initialized.
0xC0690009L	TLR_E_PROFIBUS_FSPMM2_COM_REFERENCE_INVALID Invalid communication reference.
0xC069000AL	TLR_E_PROFIBUS_FSPMM2_TIMEOUT Timeout error.
0xC069000BL	TLR_E_PROFIBUS_FSPMM2_INITIATE_ABT_STO Parameter Send Timeout to small.
0xC069000CL	TLR_E_PROFIBUS_FSPMM2_INITIATE_ABT_FE Telegram format error.
0xC069000DL	TLR_E_PROFIBUS_FSPMM2_NRS Negative response.
0xC069000EL	TLR_E_PROFIBUS_FSPMM2_ABORT Service Aborted.
0xC0690100L	TLR_E_PROFIBUS_FSPMM2_CON_XX Service confirmation negative.

Table 193: Error Messages of the FSPMM-Task

### 7.2.1 Diagnostic Codes of the FSPMM2-Task

Currently no diagnostic codes are defined for the FSPMM2-task.

### 7.3 Error Codes of the DL-Task

Hexadecimal Value	Definition Description
0xC006001L	TLR_E_PROFIBUS_DL_COMMAND_INVALID Invalid command received.
0xC0060040L	TLR_E_PROFIBUS_DL_XC_INVALID The assigned XC-Data Link Layer is not installed or has a pending error.
0xC0060041L	TLR_E_PROFIBUS_DL_BAUDRATE_INVALID The specified baudrate option is not supported and is out of range.
0xC0060042L	TLR_E_PROFIBUS_DL_GAP_UPDATE_INVALID The specified GAP update factor option is not supported and is out of range 1-100.
0xC0060043L	TLR_E_PROFIBUS_DL_DL_ADDR_INVALID The specified local Profibus address option is not supported and is out of range 0-125.
0xC0060044L	TLR_E_PROFIBUS_DL_RETRY_LIMIT The specified retry limit option is not supported and is zero.
0xC0060045L	TLR_E_PROFIBUS_DL_HSA_INVALID The specified highest station address option is not supported and is out of range 0-126.
0xC0060046L	TLR_E_PROFIBUS_DL_NO_BUS_PARAMETER_SET The service cannot be executed, there are no bus parameters specified yet.
0xC0060047L	TLR_E_PROFIBUS_DL_DLE_NOT_RESPONDING The service has detected a timeout at the connected XC-Data Link Layer entity.
0xC0060048L	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.
0xC0060049L	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.
0xC0060050L	TLR_E_PROFIBUS_DL_STOPPED Profibus is stopped command cannot be handled.
0xC0060051L	TLR_E_PROFIBUS_DL_PENDING_PACKET Previous pending packet is returned. It could not be handled.
0xC0060052L	TLR_E_PROFIBUS_DL_SLAVE_MODE Command could not be executed, DL-task is running at slave mode.
0xC0060080L	TLR_E_PROFIBUS_DL_ACK_UE The remote station the service has been sent to indicates a User Error as service acknowledgement.
0xC0060081L	TLR_E_PROFIBUS_DL_ACK_RR The remote station the service has been sent to indicates a Resource Error as service acknowledgement.
0xC0060082L	TLR_E_PROFIBUS_DL_ACK_RS The remote station the service has been sent to indicates a Service Access Point Error as service acknowledgement.
0xC0060083L	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.
0xC0060084L	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
0xC0060085L	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.
0xC0060086L	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.
0xC0060087L	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.
0xC0060088L	TLR_E_PROFIBUS_DL_ACK_NA The remote station the service has been sent to shows no or no plausible reaction at all.
0xC0060089L	TLR_E_PROFIBUS_DL_ACK_UNKNOWN The remote station the service has been sent has returned an unknown acknowledgement code.
0xC006008AL	TLR_E_PROFIBUS_DL_ACK_LS The requested service is not activated within the local SAP configuration.
0xC006008BL	TLR_E_PROFIBUS_DL_ACK_LR The local resources needed to execute the requested service are not available or not sufficient.
0xC006008CL	TLR_E_PROFIBUS_DL_ACK_DS The local data link layer is not in the logical token ring or disconnected from the network.
0xC006008DL	TLR_E_PROFIBUS_DL_ACK_IV Invalid parameter detected in the requested service.
0xC006008EL	TLR_E_PROFIBUS_DL_ACK_NO The local SAP is not activated because it has been activated already or resources are not sufficient.
0xC006008FL	TLR_E_PROFIBUS_DL_ACK_NO_SET The variable to be set does not exist.
0xC0060090L	TLR_E_PROFIBUS_DL_ACK_RE Format error of the telegram.
0xC0060091L	TLR_E_PROFIBUS_DL_TSET_INVALID The specified parameter TSET is out of range 1-255.

Table 194: Error Messages of the DL-Task

### 7.3.1 Diagnostic Codes of the DL-Task

Hexadecimal Value	Definition Description
0xC0060001L	TLR_DIAG_E_PROFIBUS_DL_DATA_ACK_RES Invalid "Data Ack (SDA)" response packet received. Response does not match to internally reserved communication block
0xC0060002L	TLR_DIAG_E_PROFIBUS_DL_DATA_RES Invalid "Data (SDN)" response packet received. Response does not match to internally reserved communication block
0xC0060003L	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 195: Diagnostic Messages of the DL-Task

## 7.4 Error Codes of the APM-Task

Hexadecimal Value	Definition Description
0xC0390003L	TLR_E_PROFIBUS_APM_TIO_RESET_W_MODE_STOP Timeout while stopping PROFIBUS.
0xC0390005L	TLR_E_PROFIBUS_APM_NON_EXCHANGE_SLAVE No data exchange with at least one slave.
0xC0390006L	TLR_E_PROFIBUS_APM_NON_EXCHANGE_ALL No slave in data exchange.
0xC0390008L	TLR_E_PROFIBUS_APM_CONFIG_VIA_DBM Already configured via data base.
0xC0390009L	TLR_E_PROFIBUS_APM_ALREADY_CONFIGURED Already configured.
0xC039000AL	TLR_E_PROFIBUS_APM_CHANNEL_INIT_IN_PROGRESS Channel initialization in progress.
0xC039000BL	TLR_E_PROFIBUS_APM_CHANNEL_INIT_FAILED Channel initialization failed.

Table 196: Error Messages of the APM-Task

### 7.4.1 Diagnostic Codes of the APM-Task

Hexadecimal Value	Definition Description
0xC0390032L	TLR_DIAG_E_PROFIBUS_APM_NO_FSPMM No FSPMM-Task or -queue found to communicate.
0xC039003AL	TLR_DIAG_E_PROFIBUS_ADR_DOUBLE Double station address detected.
0xC039003BL	TLR_DIAG_E_PROFIBUS_MAX_OUTPUT_OFFSET Maximum output offset reached.
0xC039003CL	TLR_DIAG_E_PROFIBUS_MAX_INPUT_OFFSET Maximum input offset reached
0xC039003DL	TLR_DIAG_E_PROFIBUS_OUTPUT_OVERLAP Overlap in output data.
0xC039003EL	TLR_DIAG_E_PROFIBUS_INPUT_OVERLAP Overlap in input data.
0xC03900DEL	TLR_DIAG_E_PROFIBUS_APM_AUTOCLEAR_ERROR Auto-clear activated.
0xC03900E1L	TLR_DIAG_E_PROFIBUS_APM_OUT_OF_PACKET No packet left to communicate.
0xC0390100L	TLR_DIAG_E_PROFIBUS_APM_NO_INIT_PACKET Getting a packet from the pool to initialize the Master state machine failed.
0xC0390101L	TLR_DIAG_E_PROFIBUS_APM_NO_SET_MODE_PACKET Getting a packet from the pool to set the Master mode failed.
0xC0390102L	TLR_DIAG_E_PROFIBUS_APM_NO_SET_OUTPUT_PACKET Getting a packet from the pool to set output date fail.
0xC0390103L	TLR_DIAG_E_PROFIBUS_APM_NO_GET_INPUT_PACKET Getting a packet from the pool to request input data failed.
0xC0390104L	TLR_DIAG_E_PROFIBUS_APM_GET_INPUT_FAIL Getting input data fail.
0xC0390105L	TLR_DIAG_E_PROFIBUS_APM_SET_OUTPUT_FAIL Write Slave output data failed.

Table 197: Diagnostic Messages of the APM-Task

## 8 Contact

### 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](mailto:info@hilscher.com)

#### Support

Phone: +49 (0) 6190 9907-99  
E-Mail: [de.support@hilscher.com](mailto:de.support@hilscher.com)

### Subsidiaries

#### China

Hilscher Ges.f.Systemaut. mbH  
Shanghai Representative Office  
200010 Shanghai  
Phone: +86 (0) 21-6355-5161  
E-Mail: [info@hilscher.cn](mailto:info@hilscher.cn)

#### Support

Phone: +86 (0) 21-6355-5161  
E-Mail: [cn.support@hilscher.com](mailto: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](mailto:info@hilscher.fr)

#### Support

Phone: +33 (0) 4 72 37 98 40  
E-Mail: [fr.support@hilscher.com](mailto:fr.support@hilscher.com)

#### India

Hilscher India Pvt. Ltd.  
New Delhi - 110 025  
Phone: +91 11 40515640  
E-Mail: [info@hilscher.in](mailto:info@hilscher.in)

#### Italy

Hilscher Italia srl  
20090 Vimodrone (MI)  
Phone: +39 02 25007068  
E-Mail: [info@hilscher.it](mailto:info@hilscher.it)

#### Support

Phone: +39/02 25007068  
E-Mail: [it.support@hilscher.com](mailto:it.support@hilscher.com)

#### Japan

Hilscher Japan KK  
Tokyo, 160-0022  
Phone: +81 (0) 3-5362-0521  
E-Mail: [info@hilscher.jp](mailto:info@hilscher.jp)

#### Support

Phone: +81 (0) 3-5362-0521  
E-Mail: [jp.support@hilscher.com](mailto:jp.support@hilscher.com)

#### Korea

Hilscher Korea Inc.  
Suwon-Si, 443-810  
Phone: +82-31-204-6190  
E-Mail: [info@hilscher.kr](mailto:info@hilscher.kr)

#### Switzerland

Hilscher Swiss GmbH  
4500 Solothurn  
Phone: +41 (0) 32 623 6633  
E-Mail: [info@hilscher.ch](mailto:info@hilscher.ch)

#### Support

Phone: +49 (0) 6190 9907-99  
E-Mail: [ch.support@hilscher.com](mailto:ch.support@hilscher.com)

#### USA

Hilscher North America, Inc.  
Lisle, IL 60532  
Phone: +1 630-505-5301  
E-Mail: [info@hilscher.us](mailto:info@hilscher.us)

#### Support

Phone: +1 630-505-5301  
E-Mail: [us.support@hilscher.com](mailto:us.support@hilscher.com)