



Protocol API
AS-Interface Master

V2.2.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC091201API02EN | Revision 2 | English | 2010-06 | Released | Public

Revision History

Rev	Date	Name	Revisions
1	2009-12-02	RG/AB/ES	Created
2	2010-06-07	RG	Firmware Version V2.2.1.x Rewrote section ASI_MASTER_EXECUTE_COMMAND_REQ/CNF – Execute Command (Send a single Command to the Slave) Device status added A lot of detail corrections Legal text exchanged Section Technical Data: New: Support of DMA for PCI targets and Support of slot number for CIFX 50-2ASM added

Table of Contents

1	Introduction	9
1.1	Abstract	9
1.2	Intended Audience	9
1.3	Specifications	10
1.3.1	Protocol Task System	10
1.3.2	Supported Transaction Types	10
1.3.3	Technical Data	11
1.3.4	Limitations	12
1.4	Terms, Abbreviations and Definitions	13
1.5	References	13
1.6	Legal Notes	14
1.6.1	Copyright	14
1.6.2	Important Notes	14
1.6.3	Exclusion of Liability	15
1.6.4	Export	15
2	Fundamentals	16
2.1	General Access Mechanisms on netX Systems	16
2.2	Accessing the Protocol Stack by Programming the AP Task's Queue	17
2.2.1	Getting the Receiver Task Handle of the Process Queue	17
2.2.2	Meaning of Source- and Destination-related Parameters	17
2.3	Accessing the Protocol Stack via the Dual Port Memory Interface	18
2.3.1	Communication via Mailboxes	18
2.3.2	Using Source and Destination Variables correctly	19
2.3.3	Obtaining useful Information about the Communication Channel	22
2.4	Client/Server Mechanism	24
2.4.1	Application as Client	24
2.4.2	Application as Server	24
3	Dual-Port Memory	26
3.1	Cyclic Data (Input/Output Data)	26
3.1.1	Input Data Image	27
3.1.2	Process Data Output	27
3.2	Acyclic Data (Mailboxes)	28
3.2.1	General Structure of Messages or Packets for Non-Cyclic Data Exchange	29
3.2.2	Status & Error Codes	31
3.2.3	Differences between System and Channel Mailboxes	31
3.2.4	Send Mailbox	31
3.2.5	Receive Mailbox	31
3.2.6	Channel Mailboxes (Details of Send and Receive Mailboxes)	32
3.3	Status	33
3.3.1	Common Status	33
3.3.2	Extended Status	41
3.4	Control Block	51
4	Getting started / Configuration	53
4.1	Overview about Essential Functionality	53
4.2	Configuration of Bus and Slave Parameters	54
4.2.1	Write Access to the Dual-Port Memory	54
4.2.2	Using the configuration tool SYCON.NET	54
4.2.3	Detailed Description of Bus and Master Parameters	55
4.2.4	Detailed Description of Slave Parameters	58
4.3	Task Structure of the AS-Interface Master Stack	65
5	Overview	66
5.1	Introduction to AS Interface	66
5.1.1	Data Convention	66
5.2	AS-Interface Master	67
5.2.1	General Structure of the AS-Interface Master	67
5.2.2	Execution Control	68
5.2.3	Lists maintained by Execution Control	73

5.2.4	Transmission Control	77
5.3	AS-Interface Slave -Structure and Addressing	81
5.3.1	Slave Types and Addressing Mechanisms	81
5.3.2	Slave Registers and Flags	83
5.3.3	Relation between IO Config, ID Codes and Slave Profiles.....	85
5.3.4	AS-Interface Slave States and State Machine	89
5.3.5	Tasks of the AS-Interface Slave.....	90
5.3.6	Error Handling at the Slave	90
5.4	Data Representation	91
5.4.1	Standard Slaves.....	91
5.4.2	Data Representation of Slave with up to 16 Bit Signals	91
5.4.3	Slave Profile S-0.A to S-F.A.....	93
5.5	AS-Interface and the ISO/OSI Layer Model.....	94
6	The Application Interface.....	95
6.1	The ASIMASTER -Task.....	96
6.1.1	ASI_MASTER_INITIALIZE_REQ/CNF – Initialize Master	98
6.1.2	ASI_MASTER_REGISTER_REQ/CNF – Register at ASi Master	100
6.1.3	ASI_MASTER_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle	103
6.1.4	ASI_MASTER_SET_SLAVE_PARAM_REQ/CNF – Set Slave Parameters	106
6.1.5	ASI_MASTER_GET_SLAVE_PARAM_REQ/CNF – Get Slave Parameter.....	110
6.1.6	ASI_MASTER_SET_BUS_PARAM_REQ/CNF – Set Bus Parameters	117
6.1.7	ASI_MASTER_GET_BUS_PARAM_REQ/CNF – Get Bus Parameters	121
6.1.8	ASI_MASTER_SET_OFFLINE_MODE_REQ/CNF – Set Offline Mode	123
6.1.9	ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF – Read Configuration.....	126
6.1.10	ASI_MASTER_READ_PARAMETER_IMAGE_REQ/CNF – Read Parameter Image.....	131
6.1.11	ASI_MASTER_GET_PERMANENT_CONFIG_REQ/CNF – Get Permanent Configuration	135
6.1.12	ASI_MASTER_GET_PERMANENT_PARAMETER_REQ/CNF – Get Permanent Parameter	139
6.1.13	ASI_MASTER_GET_SLAVE_DIAG_REQ/CNF – Get Slave Diagnosis	143
6.1.14	ASI_MASTER_EXECUTE_COMMAND_REQ/CNF – Execute Command (Send a single Command to the Slave).....	147
6.1.15	ASI_MASTER_WRITE_PARAMETER_REQ/CNF – Write Parameter to Slave	151
6.1.16	ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ/CNF – Change Slave Address	155
6.1.17	ASI_MASTER_WRITE_EXT_ID1_CODE_REQ/CNF – Write Extended ID 1 Code	158
6.1.18	ASI_MASTER_GET_LPS_REQ /CNF – Get List of Projected Slaves	161
6.1.19	ASI_MASTER_GET_LAS_REQ/CNF – Get List of Activated Slaves.....	164
6.1.20	ASI_MASTER_GET_LDS_REQ/CNF – Get List of Detected Slaves	167
6.1.21	ASI_MASTER_GET_LPF_REQ/CNF – Get List of Peripheral Faults	170
6.1.22	ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_REQ/CNF – Get Execution Control Flags 173	
6.1.23	ASI_MASTER_READ_IDENTIFICATION_STRING_REQ/CNF – Read Identification String....	178
6.1.24	ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ/CNF – Read Diagnostic String.....	185
6.1.25	ASI_MASTER_READ_PARAMETER_STRING_REQ/CNF – Read Parameter String.....	190
6.1.26	ASI_MASTER_WRITE_PARAMETER_STRING_REQ/CNF – Write Parameter String	195
6.1.27	ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode	198
6.1.28	ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF – Set Data Exchange Active.....	201
6.1.29	ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ/CNF – Set Auto Address Enable	204
6.1.30	ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_REQ/CNF – Get Auto Address Enable.....	207
6.1.31	ASI_MASTER_STATE_CHANGE_IND/RES – State Change Indication.....	210
6.2	The ASIAPM -Task	213
6.2.1	ASI_APM_GET_STATE_REQ/CNF – Get State	214
6.2.2	ASI_APM_STORE_ACTUAL_CONFIG_REQ/CNF – Store Actual Configuration	216
6.2.3	ASI_APM_STORE_ACTUAL_PARAM_REQ/CNF – Store Actual Parameters	219
6.2.4	ASI_APM_SET_LPS_REQ/CNF – Set List of Projected Slaves.....	221
7	Status/Error Codes Overview.....	224
7.1	Status/Error Codes ASIMASTER-Task	224
7.2	Status/Error codes APM-Task.....	226
8	Contact.....	227

List of Figures

Figure 1: The three different Ways to access a Protocol Stack running on a netX System	16
Figure 2 - Use of ulDest in Channel and System Mailbox	19
Figure 3 - Using ulSrc and ulSrcId.....	20
Figure 4: Transition Chart Application as Client.....	24
Figure 5: Transition Chart Application as Server.....	25
Figure 6: Task Structure of the AS-Interface Master Stack:.....	65
Figure 7: Identification String (Read)	178

List of Tables

Table 1: Names of Tasks in EtherNet/IP Firmware	10
Table 2: Terms, Abbreviations and Definitions	13
Table 3: References	13
Table 4: Names of Queues in EtherNet/IP Firmware	17
Table 5: Meaning of Source- and Destination-related Parameters.	17
Table 6: Meaning of Destination-Parameter ulDest.Parameters	19
Table 7 Example for correct Use of Source- and Destination-related parameters.:	21
Table 8: Input Data Image:	27
Table 9: Output Data Image	27
Table 10: Channel Mailboxes	32
Table 11: Common Status Structure Definition	34
Table 12: Communication State of Change	35
Table 13: Meaning of Communication Change of State Flags	36
Table 14: Master Status Structure Definition	39
Table 15: rcX Status and Error Codes/Slave	40
Table 16: Extended Status Block	41
Table 17: Extended Status Block Structure Definition for AS-Interface Master.....	43
Table 18: Contents of Global Flags Variable	43
Table 19: Error Types in Global Bits	44
Table 20: Operation Modes of the AS-Interface Master and their associated Values	44
Table 21: Contents of ASI Flags Variable	45
Table 22: Meaning of ASI flags	46
Table 23: Relationship between Slave Station Address and the corresponding bit in abProjectedList	46
Table 24: Relationship between Slave Station Address and the corresponding bit in abActivatedList.	47
Table 25: Relationship between Slave Station Address and the corresponding bit in abDiagnosticList	47
Table 26: Relationship between Slave Station Address and the corresponding bit in abDetectedList.....	48
Table 27: Relationship between Slave Station Address and the corresponding bit in abPeripheryFaultList	49
Table 28: Relationship between Slave Station Address and the corresponding bit in abPeripheryFaultList – Channel 1	49
Table 29: Extended Status Block for AS-interface-Master – Second part (State Field Definition Block).....	51
Table 30: Communication Control Block.....	52
Table 31: Overview about Essential Functionality (Cyclic and acyclic Data Transfer and Alarm Handling).....	53
Table 32: Bus and Master Parameters, their Meanings and their Ranges of allowed Values	55
Table 33: Possible Values of bAction and their Meanings	59
Table 34: Slave Parameters, their Meanings and their Ranges of allowed Values	59
Table 35: Meaning of bDevFlag	60
Table 36: Meaning of bVersionFlags	61
Table 37: Meaning of bConfigFlags	61
Table 38: Example of data type description part for an I/O device with device profile S-3.0 (2 in/2 out).....	63
Table 39: Example of data type description part for a device with the profile S-7.3 (here: analog input module, 2 channels).....	63
Table 40: Example of configuration offset part for an I/O device with device profile S-3.0 (2 in/2 out).....	64
Table 41: Example of configuration offset part for a device with the profile S-7.3 (here: analog input module, 2 channels).....	64
Table 42: Relation between Execution Control Functions and the corresponding Packets	72
Table 43: Types of Single Transactions in AS-Interface	78
Table 44: Available Combined Transaction Types.....	79
Table 45: AS-Interface Slave Addresses in the Application Interface	82
Table 46: The Status Register of the AS-Interface Slave.....	84
Table 47: Relation between Slave Profile and IO-Configuration and the ID-Codes for Standard Slaves.....	85
Table 48: Relation between Slave Profile and IO-Configuration and the ID-Codes for Slaves supporting Extended Addressing according to AS-interface Specification 2.1	86
Table 49: Explanation of Abbreviations in Table 47 and Table 48	86
Table 50: Lists of possible Profiles for Standard Slaves	87
Table 51: Lists of possible Profiles for Slaves supporting Extended Addressing	88
Table 52: ASIMASTER -Task Process Queue	96
Table 53: Topics of ASIMASTER -Task and associated packets	97
Table 54: ASI_MASTER_PACKET_INITIALIZE_REQ_T – Initialize Master	98
Table 55: ASI_MASTER_PACKET_INITIALIZE_CNF_T –Confirmation for Initialize Master	99
Table 56: ASI_MASTER_PACKET_APP_REGISTER_REQ_T – Register at ASi Master Request.....	101
Table 57: ASI_MASTER_PACKET_APP_REGISTER_CNF_T –Confirmation for Register at ASi Master Request	102
Table 58: ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_T – Get Buffer Handle Request.....	103

Table 59: ASI_MASTER_PACKET_GET_BUFFER_HANDLE_CNF_T –Confirmation for Get Buffer Handle Request.....	105
Table 60: ASI_MASTER_PACKET_SET_SLAVE_PARAM_REQ_T – Set Slave Parameters	108
Table 61: ASI_MASTER_PACKET_SET_SLAVE_PARAM_CNF_T – Confirmation for Set Slave Parameters	109
Table 62: Structure ASI_MASTER_SLAVE_PARAM_T.....	110
Table 63: Meaning of allowed Values of variable bAction.....	110
Table 64: Structure ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T.....	111
Table 65: Structure of ausTypeData Entries.....	111
Table 66: Structure ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T.....	112
Table 67: Structure of ausIoOffset Entries.....	112
Table 68: ASI_MASTER_PACKET_GET_SLAVE_PARAM_REQ_T - Get Slave Parameter	113
Table 69: ASI_MASTER_PACKET_GET_SLAVE_PARAM_CNF_T - Confirmation for Get Slave Parameter	116
Table 70: ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_T – Set Bus Parameters	119
Table 71: ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_T –Confirmation for Set Bus Parameters	120
Table 72: ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_T – Get Bus Parameters	121
Table 73: ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_T – Confirmation for Get Bus Parameters.....	122
Table 74: ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_T – Set Offline Mode	124
Table 75: ASI_MASTER_PACKET_SET_OFFLINE_MODE_CNF_T – Confirmation for Set Offline Mode	125
Table 76: Configuration Data Structure ASI_MASTER_SLAVE_CONFIG_DATA_T	126
Table 77: ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_T – Read Configuration.....	128
Table 78: ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_CNF_T –Confirmation for Read Configuration.....	130
Table 79: ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_T –Read Parameter Image	132
Table 80: ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_CNF_T –Confirmation for Read Parameter Image	134
Table 81: Configuration Data Structure ASI_MASTER_SLAVE_CONFIG_DATA_T	135
Table 82: ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_T – Get Permanent Configuration Request	136
Table 83: ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_T –Confirmation for Get Permanent Configuration Request.....	138
Table 84: ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_T – Get Permanent Parameter	140
Table 85: ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_CNF_T – Confirmation for Get Permanent Parameter.....	142
Table 86: Data structure ASI_MASTER_SLAVE_DIAG_T	143
Table 87: ASI_MASTER_GET_SLAVE_DIAG_REQ_T – Get Slave Diagnosis	144
Table 88: ASI_MASTER_PACKET_GET_SLAVE_DIAG_CNF_T –Confirmation for Get Slave Diagnosis.....	146
Table 89: Possible Command Values for Parameter ulInfo	147
Table 90: ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_T – Execute Command Request	149
Table 91: ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_T – Confirmation for Execute Command Request	150
Table 92: ASI_MASTER_WRITE_PARAMETER_REQ_T – Write Parameter to Slave Request	152
Table 93: ASI_MASTER_WRITE_PARAMETER_REQ_T – Confirmation for Write Parameter to Slave Request....	154
Table 94: ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T – Change Slave Address Request.....	156
Table 95: ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_CNF_T –Confirmation for Change Slave Address Request.....	157
Table 96: ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_T – Write External ID 1 Code	159
Table 97: ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_CNF_T – Confirmation for Write External ID 1 Code Request.....	160
Table 98: Representation of List of Projected Slaves (LPS)	161
Table 99: ASI_MASTER_PACKET_GET_LPS_REQ_T – Get List of projected Slaves Request	162
Table 100: ASI_MASTER_PACKET_GET_LPS_CNF_T – Confirmation for Get List of projected Slaves Request.....	163
Table 101: Representation of List of Activated Slaves (LAS).....	164
Table 102: ASI_MASTER_PACKET_GET_LAS_REQ_T – Get List of Activated Slaves Request	165
Table 103: ASI_MASTER_PACKET_GET_LAS_CNF_T –Confirmation for Get List of Activated Slaves Request	166
Table 104: Representation of List of Detected Slaves (LDS).....	167
Table 105: ASI_MASTER_PACKET_GET_LDS_REQ_T – Get List of Detected Slaves Request.....	168
Table 106: ASI_MASTER_PACKET_GET_LDS_CNF_T –Confirmation for Get List of Detected Slaves Request.....	169
Table 107: Representation of List of Peripheral Faults (LPF)	170
Table 108: ASI_MASTER_PACKET_GET_LPF_REQ_T – Get List of Peripheral Faults Request.....	171
Table 109: ASI_MASTER_PACKET_GET_LPF_CNF_T –Confirmation for Get List of Peripheral Faults Request.....	172
Table 110: Meaning of ASI Flags.....	174
Table 111: ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_REQ_T – Get Execution Control Flags Request.....	175
Table 112: ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_CNF_T – Confirmation for Get Execution Control Flags Request.....	177

Table 113: Contents of ID String Triple 1, Byte 0.....	179
Table 114: Contents of ID String Triple 1, Byte 1.....	179
Table 115: Contents of ID String Triple 1, Byte 2.....	180
Table 116: Contents of ID String Triple 2, Byte 3.....	180
Table 117: Contents of ID String Triple 2, Byte 4.....	180
Table 118: Contents of ID String Triple 2, Byte 5.....	180
Table 119: Contents of ID String Triple 3, Byte 6.....	180
Table 120: Contents of ID String Triple 3, Byte 7.....	181
Table 121: Contents of ID String Triple 3, Byte 8.....	181
Table 122: ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_T – Read Identification String Request.....	182
Table 123: ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_CNF_T – Confirmation for Read Identification String Request.....	184
Table 124: Contents of Diagnostic String Triple 1, Byte 0.....	185
Table 125: Contents of Diagnostic String Triple 1, Byte 1.....	185
Table 126: Contents of Diagnostic String Triple 1, Byte 2.....	185
Table 127: ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_T – Read Diagnostic String Request.....	187
Table 128: ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_CNF_T –Confirmation for Read Diagnostic String Request.....	189
Table 129: Contents of Diagnostic String Triple 1, Byte 0.....	190
Table 130: Contents of Diagnostic String Triple 1, Byte 1.....	190
Table 131: Contents of Diagnostic String Triple 1, Byte 2.....	190
Table 132: ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_T – Read Parameter String Request.....	192
Table 133: ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNF_T – Confirmation for Read Parameter String Request.....	194
Table 134: ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_T – Write Parameter String Request.....	196
Table 135: ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_CNF_T –Confirmation for Write Parameter String Request.....	197
Table 136: ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_T – Set Operation Mode Request.....	199
Table 137: ASI_MASTER_PACKET_SET_OPERATION_MODE_CNF_T –Confirmation for Set Operation Mode Request.....	200
Table 138: ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_T – Set Data Exchange Active Request.....	202
Table 139: ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_CNF_T – Confirmation for Set Data Exchange Active Request.....	203
Table 140: ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_T – Set Auto Address Enable Request.....	205
Table 141: ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_CNF_T – Confirmation for Set Auto Address Enable Request.....	206
Table 142: ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_T – Get Auto Address Enable Request.....	207
Table 143: ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_CNF_T – Confirmation for Get Auto Address Enable Request.....	209
Table 144: ASI_MASTER_PACKET_STATE_CHANGE_IND_T – State Change Indication.....	211
Table 145: ASI_MASTER_PACKET_STATE_CHANGE_RES_T – Response to State Change Indication.....	212
Table 146: ASIAPM -Task process queue.....	213
Table 147: Topics of ASIAPM -Task and associated packets.....	213
Table 148: ASI_APM_PCK_GET_STATE_REQ_T – Get State Request.....	214
Table 149: ASI_APM_PCK_GET_STATE_CNF_T –Confirmation for Get State Request.....	215
Table 150: ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_T – Store Actual Configuration Request.....	217
Table 151: ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_T – Confirmation for Store Actual Configuration Request.....	218
Table 152: ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_T – Store Actual Parameters Request.....	219
Table 153: ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_T – Confirmation for Store Actual Parameters Request.....	220
Table 154: Representation of List of Projected Slaves (LPS).....	221
Table 155: ASI_APM_PCK_SET_LPS_REQ_T – Set List of Projected Slaves Request.....	222
Table 156: ASI_APM_PCK_SET_LPS_CNF_T –Confirmation for Set List of Projected Slaves Request.....	223
Table 157: Status/Error Codes ASIMASTER -Task.....	225
Table 158: Status/Error Codes APM-Task.....	226

1 Introduction

1.1 Abstract

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

Stack development is based on Hilscher's Task Layer Reference Programming Model. This model defines the general template used to create a task including a combination of appropriate functions belonging to the same type of protocol layer. Furthermore, it defines of how different tasks have to communicate with each other in order to exchange data between each communication layer. This Reference Model is used by all programmers at Hilscher and shall be used by the developer when writing an application task on top of the stack.

System Requirements

This software package has the following environmental system requirements:

- netX-Chip as CPU hardware platform
- Operating system for task scheduling required

1.2 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 AS-Interface V3.0 Specification

1.3 Specifications

Protocol Task System

To manage the AS-Interface Master implementation x tasks are involved into the system. To send packets to a task, the task main queue have to be identifier. For the identifier for the tasks and there Queues are the following naming conversion:

Task Name	Queue Name	Description
"QUE_ASIMASTER"	"QUE_ASIMASTER"	Name of the ASI Master Task process queue
"QUE_ASIAPM"	"QUE_ASIAPM"	Name of the ASI APM-Task process queue

Table 1: Names of Tasks in EtherNet/IP Firmware

1.3.2 Supported Transaction Types

- Standard input and output cyclic data exchange
- Combined transaction type 1: Half-duplex data transfer of bytes
- Combined transaction type 2: Full-duplex bit-serial data transfer
- Combined transaction type 3: Full-duplex 4 bit or 8 bit data transfer
- Combined transaction type 4: Single or dual 16 bit data transfer in extended addressing mode
- Combined transaction type 5: Full-duplex 16 bit fast data transfer

1.3.3 Technical Data

The data below applies to AS-Interface Master firmware and stack version 2.0.x

Technical Data

Maximum number of supported slaves	Max. 62 slaves
Maximum number of total cyclic input data	Max. 248 bits using digital slaves Max. 248 bytes using analog (transparent) slaves The maximum number depends on the used slave profiles
Maximum number of total cyclic output data	Max. 248 bits using digital slaves Max. 248 bytes using analog (transparent) slaves The maximum number depends on the used slave profiles
Maximum number of cyclic input data	Max. 4 Bit digital data Max. 4 channel with up to 16 bit analog data The maximum number depends on the used slave profiles
Maximum number of cyclic output data	Max. 4 Bit digital data Max. 4 channel with up to 16 bit analog data The maximum number depends on the used slave profiles
Parameterization data	4 bit per standard slave 3 bit per extended slave
Maximum number of acyclic read/write	Max. 220 bytes for string transfer
Functions	Support of data exchange via combined transaction types 1, 2, 3, 4 and 5 (CTT 1-5) Automatic address assignment Modification of address and Extended ID1-Code of Slave supported Profile for extended Master: M4
Baud rate	166,67 kBaud
AS-Interface specification	3.0 Revision 2
Firmware/stack available for netX	
netX 50	no
netX 100, netX 500	yes

1.4 Terms, Abbreviations and Definitions

Term	Description
AP	Application on top of the stack
ECTRL	Execution control
APF	AS-Interface power failure
APO	AS-Interface power on
CDI	Configuration data image
PCD	Permanent configuration data
PI	Parameter image
PP	Permanent parameter
LDS	List of detected Slaves
LAS	List of activated Slaves
LPS	List of projected Slaves
LPF	List of peripheral faults
IDI	Input data image
ODI	Output data image
AIDI	Analog input data image
AODI	Analog output data image

Table 2: Terms, Abbreviations and Definitions

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

1.5 References

This document is based on the following specifications:

1	netX DPM Interface Manual, Hilscher GmbH
2	Task Layer Reference Manual, Hilscher GmbH
3	Actuator Sensor Interface Complete Specification Version 3.0, Revision 0, AS-International Association
4	Actuator Sensor Interface Profiles (Annex A and B) Version 3.0, Revision 0, AS-International Association

Table 3: References

1.6 Legal Notes

1.6.1 Copyright

© 2009-2010 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.

1.6.2 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.

1.6.3 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.

1.6.4 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:

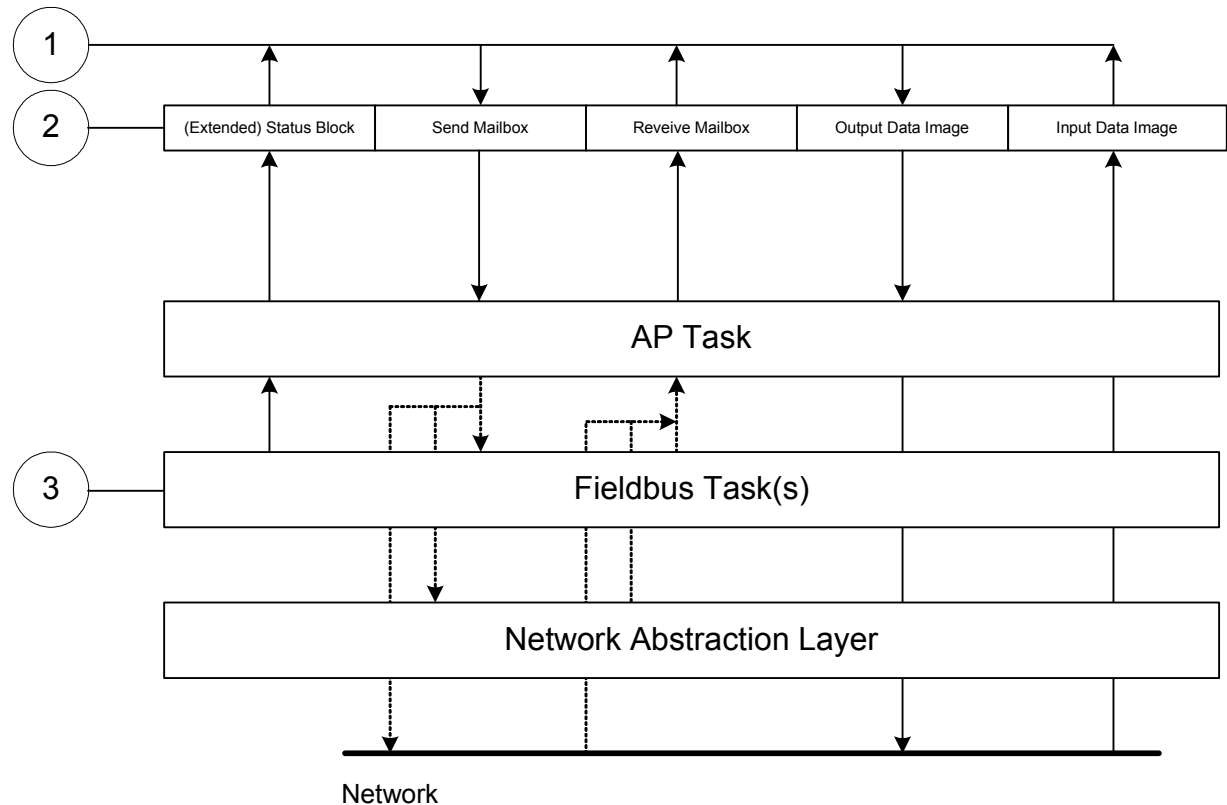


Figure 1: The three different Ways to access a Protocol Stack running on a netX System

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 shared 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 the `QUE_ASIMASTER`-Task or the `QUE_ASIAPM`-Task 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 names for accessing the `QUE_ASIMASTER`-Task or the `QUE_ASIAPM`-Task, which you have to use as current value for the first parameter (`pszIdn`), is

ASCII Queue name	Description
"QUE_ASIMASTER"	Name of the ASI Master Task process queue
"QUE_ASIAPM"	Name of the ASI APM-Task process queue

Table 4: Names of Queues in EtherNet/IP Firmware

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the `QUE_ASIMASTER`-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

Table 5: Meaning of Source- and Destination-related Parameters.

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. Chapter 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 EtherNet/IP-Adapter 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. [Acyclic Data \(Mailboxes\)](#) 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:

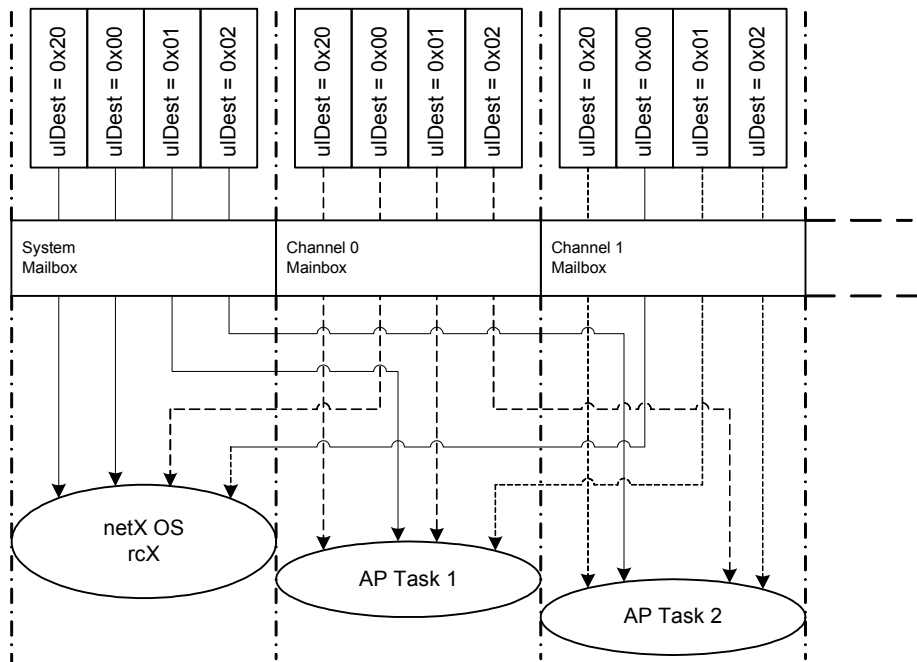


Figure 2 - Use of `ulDest` in Channel and System Mailbox

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

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

Table 6: Meaning of Destination-Parameter `ulDest`.Parameters.

The figure 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 figure:

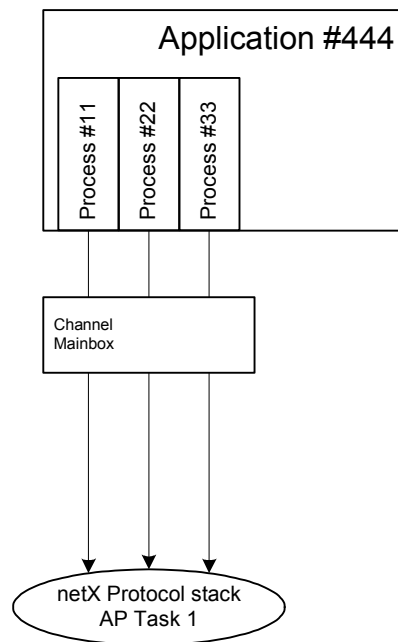


Figure 3 - Using `ulSrc` and `ulSrcId`

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	ulDest	= 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	ulSrc	= 444	Denotes the host application (#444).
Destination Identifier	ulDestId	= 0	In this example, it is not necessary to use the destination identifier.
Source Identifier	ulSrcId	= 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.

Table 7 Example for correct Use of Source- and Destination-related parameters.:

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_AS-INTERFACE = 0x0020`

If true, this denotes that this xCPort is suitable for running the AS-Interface Master 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.

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).

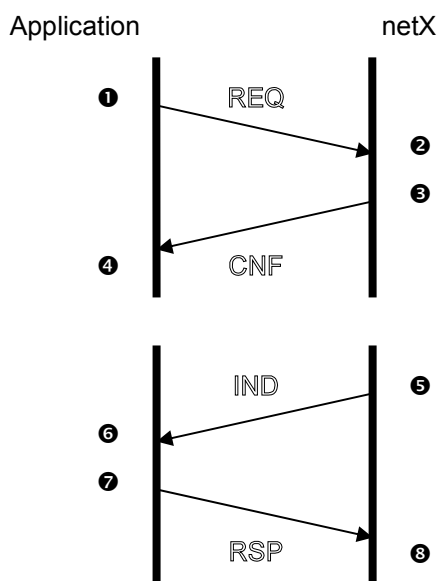


Figure 4: Transition Chart Application as Client

- 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).

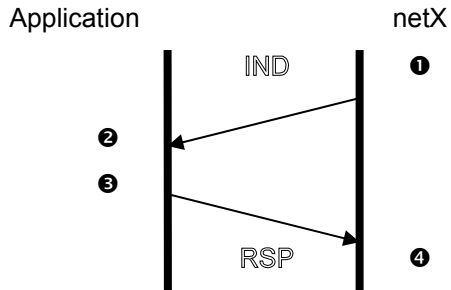


Figure 5: Transition Chart Application as Server

- ❶ ❷ The netX firmware passes an indication packet through the mailbox.
- ❸ ❹ 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: 24 byte are used for status information (8 byte for list of configured slaves, 8 byte for list of activated slaves and 8 byte for list of slaves with faults or errors).

The contents of these 24 byte is identical to the contents of the second part of the Extended Status Block beginning at address 0x0100, see *Table 29: Extended Status Block for AS-Interface-Master – Second part (State Field Definition Block)* of this document.

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

Table 8: Input Data Image:

Note: In case of a network fault (e.g. 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. 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 9: 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*:

For a list of codes see the *netX Dual-Port Memory Interface Manual*.

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 from the network or from the protocol stack

Table 10: 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
0x0028	UINT32	ulErrorCount	<u>Error Count</u> Total Number of Detected Error Since Power-Up or Reset
0x002C	UINT32	ulErrorLogInd	<u>Error Log Indicator</u> Total Number Of Entries In The Error Log Structure (not supported yet)
0x0030	UINT32	ulReserved[2]	<u>Reserved</u> Set to 0

Table 11: 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 12: 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 41).
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 13: 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 14: Master Status Structure Definition

```
typedef struct tagNETX_MASTER_STATUS
{
    UINT32 ulSlaveState;
    UINT32 ulSlaveErrLogInd;
    UINT32 ulNumOfConfigSlaves;
    UINT32 ulNumOfActiveSlaves;
    UINT32 ulNumOfDiagSlaves;
    UINT32 ulReserved;
} NETX_MASTER_STATUS;
```

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 15: 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 *SYCON.net* 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*).

Note: Please have in mind, that all offsets mentioned in this section are relative to the beginning of the common status block, as the start offset of this block depends on the size and location of the preceding blocks.

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 96 bytes correspond to ASI_MASTER_EXTENDED_STATE_T)	abExtendedStatus[172]	Area containing DeviceNet-related information. See <i>Table 17: Extended Status Block Structure Definition for AS-Interface Master below</i> (first 96 bytes of abExtendedStatus)
0x00B0	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 16: 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 the AS-Interface Master protocol implementation is stored at the address **0x0050** and structured as follows:

```
typedef struct ASI_MASTER_EXTENDED_STATE_Ttag
    ASI_MASTER_EXTENDED_STATE_T;

#define ASI_MASTER_LIST_SIZE 8

#define ASI_MASTER_GLOBAL_FLAGS_CONTROL 0x01
#define ASI_MASTER_GLOBAL_FLAGS_AUTO_CLEAR 0x02
#define ASI_MASTER_GLOBAL_FLAGS_NONE_EXCHANGE 0x04
#define ASI_MASTER_GLOBAL_FLAGS_NOT_READY 0x20
#define ASI_MASTER_GLOBAL_FLAGS_INACTIVE 0x40
#define ASI_MASTER_GLOBAL_FLAGS_TRANSPARENT_MODE 0x80

#define ASI_MASTER_MASTER_STATE_OFFLINE 0x00
#define ASI_MASTER_MASTER_STATE_STOP 0x40
#define ASI_MASTER_MASTER_STATE_CLEAR 0x80
#define ASI_MASTER_MASTER_STATE_OPERATE 0xC0

#define ASI_MASTER_ASI_FLAGS_CONFIG_OK 0x0001
#define ASI_MASTER_ASI_FLAGS_LDS0 0x0002
#define ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_ASSIGN 0x0004
#define ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_AVAILABLE 0x0008
#define ASI_MASTER_ASI_FLAGS_CONFIGURATION_ACTIVE 0x0010
#define ASI_MASTER_ASI_FLAGS_NORMAL_OPERATION_ACTIVE 0x0020
#define ASI_MASTER_ASI_FLAGS_APF_NOT_APO 0x0040
#define ASI_MASTER_ASI_FLAGS_OFFLINE_READY 0x0080
#define ASI_MASTER_ASI_FLAGS_PERIPHERY_OK 0x0100

struct ASI_MASTER_EXTENDED_STATE_Ttag
{
    TLR_UINT8 bGlobalFlags;
    TLR_UINT8 bMasterState;
    TLR_UINT8 abReserved1[6];
    TLR_UINT16 usAsiFlags;
    TLR_UINT8 abReserved2[6];
    TLR_UINT8 abProjectedList[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abReserved3[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abActivatedList[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abReserved4[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abDiagnosticList[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abReserved5[ASI_MASTER_LIST_SIZE];

    TLR_UINT8 abDetectedList[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abReserved6[ASI_MASTER_LIST_SIZE];

    TLR_UINT8 abPeripheryFaultList[ASI_MASTER_LIST_SIZE];
    TLR_UINT8 abReserved7[ASI_MASTER_LIST_SIZE];
};
```

Extended Status Block for AS-Interface Master			
Address	Type	Name	Description
0x50	UINT8	bGlobalFlags	Global bits (Bit field to show bus and master main errors)
0x51	UINT8	bMasterState	Master main state (see below)
0x52	UINT8[6]	abReserved1	Reserved area
0x58	UINT16	usAsiFlags	ASI flags
0x5A	UINT8[6]	abReserved2	Reserved area
0x60	UINT8[8]	abProjectedList	List of projected slaves
0x68	UINT8[8]	abReserved3	Reserved area
0x70	UINT8[8]	abActivatedList	List of activated slaves
0x78	UINT8[8]	abReserved4	Reserved area
0x80	UINT8[8]	abDiagnosticList	List of diagnostic slaves
0x88	UINT8[8]	abReserved5	Reserved area
0x90	UINT8[8]	abDetectedList	List of detected slaves
0x98	UINT8[8]	abReserved6	Reserved area
0xA0	UINT8[8]	abPeripheryFaultList	Periphery fault list
0xA8	UINT8[8]	abReserved7	Reserved area

Table 17: Extended Status Block Structure Definition for AS-Interface Master

The single items of the Extended Status Block for AS-Interface Master have the following meaning:

■ bGlobalFlags / Global Flags

The global flags byte bGlobalFlags 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	Bit 0
ASI_MAS	ASI_MAS	ASI_MAS	Unused	Unused	ASI_MAS	ASI_MAS	ASI_MAS
TER_GLO	TER_GLO	TER_GLO			TER_GLO	TER_GLO	TER_GLO
BAL_FLA	BAL_FLA	BAL_FLA			BAL_FLA	BAL_FLA	BAL_FLA
GS_TRAN	GS_INAC	GS_NOT_			GS_NONE	GS_AUTO	GS_CONT
SPARENT	TIVE	READY			_EXCHAN	_CLEAR	ROL
_MODE					GE		

Table 18: Contents of Global Flags Variable

This bit field serves as collective display of global notifications. Notified errors can either occur at the device itself or at the slaves.

In detail these bits have the following meaning:

Bit	Error type	Explanation
0	ASI_MASTER_GLOBAL_FLAGS_CONTROL	CONTROL-ERROR: This error is caused by incorrect parameterization.
1	ASI_MASTER_GLOBAL_FLAGS_AUTO_CLEAR	AUTO-CLEAR-ERROR: The device stopped the communication to all slaves and reached the auto-clear end state
2	ASI_MASTER_GLOBAL_FLAGS_NONE_EXCHANGE	NON-EXCHANGE-ERROR: At least one slave has not reached the data exchange state and no process data are exchanged with it.
5	ASI_MASTER_GLOBAL_FLAGS_NOT_READY	Host-NOT-READY-NOTIFICATION: Indicates if the host program has set its state to operative or not. If this bit is set the host program is not ready to communicate
6	ASI_MASTER_GLOBAL_FLAGS_INACTIVE	INACTIVITY-ERROR: The Device has detected an overstepped timeout supervision time because of rejected AS Interface telegrams. It's an indication for bus short circuits while the master interrupts the communication. The bit will be set when the first timeout was detected and will not be deleted any more.
7	ASI_MASTER_GLOBAL_FLAGS_TRANSPARENT_MODE	TRANSPARENT MODE: The Device is in transparent mode.

Table 19: Error Types in Global Bits

■ bMasterState / Master main state

The master main state represents the operation mode of the AS-Interface Master stack. This operation mode is defined in section “ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode” of this document. Allowed operation modes are:

Operation mode	Value
ASI_MASTER_MASTER_STATE_OFFLINE	0x00
ASI_MASTER_MASTER_STATE_STOP	0x40
ASI_MASTER_MASTER_STATE_CLEAR	0x80
ASI_MASTER_MASTER_STATE_OPERATE	0xC0

Table 20: Operation Modes of the AS-Interface Master and their associated Values

Changes of the master main state are indicated by the *state change indication*, also see section “ ASI_MASTER_STATE_CHANGE_IND/RES – State Change Indication” of this document for more information. If you want to change this state, you can accomplish this by sending a *Set Operation Mode* packet to the ASIMASTER task, also see section “ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode” of this document.

■ usAsiFlags / ASI Flags

These flags signal the state of the AS-Interface Master for the related channel to the user application and allows the application to control the Master.

The ASI flags word `usAsiFlags` is a bit field containing the following information:

Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A	ASI_MA STER_A SI_FL A
GS_PER IPHERY _OK	GS_OFF LINE_R EADY	GS_APF _NOT_A PO	GS_NOR MAL_OP ERATIO N_ACTI VE	GS_CON FIGURA TION_A CTIVE	GS_AUT O_ADDR ESS_AV AILABL E	GS_AUT O_ADDR ESS_AS SIGN	GS_LDS 0	GS_CON FIG_OK

Table 21: Contents of ASI Flags Variable

Bits 9 to 15 are currently not used and reserved for later use.

In detail these bits have the following meaning:

Bit	Flag	Explanation
0	ASI_MASTER_ASI_FLAGS_CONFIG_OK	CONFIGURATION OK: This flag signals that the configuration is correct..
1	ASI_MASTER_ASI_FLAGS_LDS0	LDS0 (Slave with Zero Address Detected): This flag signals that a slave with zero address has been detected. If set, a slave addressed with zero is detected. LDS.0 represents the accompanying bit of the List of Detected Slaves (LDS).
2	ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_ASSIGN	AUTO_ADDRESS_ASSIGN: This flag signals that automatic address assignment is possible. If set, Automatic Address Assignment is possible.
3	ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_AVAILABLE	AUTO_ADDRESS_AVAILABLE: This flag signals that automatic address assignment is possible. If set, Automatic Address Assignment will be processed as soon as a slave addressed with zero and valid configuration data occurred.
4	ASI_MASTER_ASI_FLAGS_CONFIGURATION_ACTIVE	CONFIGURATION_ACTIVE: This flag signals that the configuration mode is active.

		If set, the Master is in configuration mode.
5	ASI_MASTER_ASI_FLAGS_NORMAL_OPERATION_ACTIVE	NORMAL_OPERATION_ACTIVE: This flag signals that normal operation is active. If set, the Master is in normal operation mode.
6	ASI_MASTER_ASI_FLAGS_APF_NOT_APO	APF_NOT_APO: This flag signals an ASi power failure. If set, AS-Interface system power is low or power down occurred during data transmission.
7	ASI_MASTER_ASI_FLAGS_OFFLINE_READY	OFFLINE_READY: Offline ready If set, the Master is in offline phase.
8	ASI_MASTER_ASI_FLAGS_PERIPHERY_OK	PERIPHERY OK: This flag signals that no peripheral fault has been detected. If set, no peripheral fault has been detected (all entries in the LPF are 0).

Table 22: Meaning of ASI flags

■ `abProjectedList` / List of projected slaves

This field consists of 8 bytes and contains the List of Projected Slaves (LPS) as a bit field for the corresponding channel. The following table shows, which bit is related to which slave.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x0	7	6	5	4	3	2	1	0
0x1	15	14	13	12	11	10	9	8
0x2	23	22	21	20	19	18	17	16
...								
0x7	x	62	61	60	59	58	57	56

Table 23: Relationship between Slave Station Address and the corresponding bit in `abProjectedList`

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means does not matter.

■ `abActivatedList` / List of activated slaves

This field consists of 8 bytes and contains the List of Activated Slaves (LAS) as a bit field for the corresponding channel. The following table shows, which bit is related to which slave.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x0	7	6	5	4	3	2	1	0
0x1	15	14	13	12	11	10	9	8
0x2	23	22	21	20	19	18	17	16
...								
0x7	x	62	61	60	59	58	57	56

Table 24: Relationship between Slave Station Address and the corresponding bit in `abActivatedList`.

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means does not matter.

■ `abDiagnosticList` / List of slaves with diagnostic information

This field consists of 8 bytes and contains the diagnostic bit as a bit field for the corresponding channel. The following table shows, which bit is related to which slave.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x0	7	6	5	4	3	2	1	0
0x1	15	14	13	12	11	10	9	8
0x2	23	22	21	20	19	18	17	16
...								
0x7	x	62	61	60	59	58	57	56

Table 25: Relationship between Slave Station Address and the corresponding bit in `abDiagnosticList`.

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means does not matter.

■ `abDetectedList` / List of detected slaves

This field consists of 8 bytes and contains the *List of Detected Slaves* (LDS) as a bit field for the corresponding channel. The following table shows, which bit is related to which slave.

	Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset									
0x0		7	6	5	4	3	2	1	0
0x1		15	14	13	12	11	10	9	8
0x2		23	22	21	20	19	18	17	16
...									
0x7		x	62	61	60	59	58	57	56

Table 26: Relationship between Slave Station Address and the corresponding bit in `abDetectedList`

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means does not matter.

■ `abPeripheryFaultList` / Periphery fault list

This field consists of 8 bytes and contains the list of slaves reporting a periphery fault (LPF) as a bit field for the corresponding channel. The following table shows, which bit is related to which slave of channel 0.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0x0	7	6	5	4	3	2	1	0
0x1	15	14	13	12	11	10	9	8
0x2	23	22	21	20	19	18	17	16
...								
0x7	x	62	61	60	59	58	57	56

Table 27: Relationship between Slave Station Address and the corresponding bit in `abPeripheryFaultList`

Similarly, the following table applies for channel 1 if present:

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Offset								
0xA8	7	6	5	4	3	2	1	0
0xA9	15	14	13	12	11	10	9	8
0xAA	23	22	21	20	19	18	17	16
...								
0xAF	x	62	61	60	59	58	57	56

Table 28: Relationship between Slave Station Address and the corresponding bit in `abPeripheryFaultList - Channel 1`

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means does not matter.

Besides the global flags, the 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, 0x80, 0x90 and 0xA0 (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 AS-interface-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]	Structure to define State field properties
0x0100	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=1. Corresponds to a bit list (one bit per 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 list of projected slaves. See description of the list of projected slaves above.
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[1]	Structure to define State field properties
0x0108	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=2. Corresponds to a bit list (one bit per 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 list of activated slaves. See description of the list of activated slaves above.
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[2]	Structure to define State field properties
0x0110	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=3. Corresponds to a bit list (one bit per 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 list of diagnostic slaves. See description of the list of diagnostic slaves above
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[3]	Structure to define State field properties
0x0118	unsigned char	bStateArea	=0. State field is located in standard input area of

			channel 0
	unsigned char	bStateTypeID	=4. Corresponds to a bit list (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 list of detected slaves See description of the list of detected slaves above.
↓	NETX_EXTENDED_STATE_STRUCT_T	atStateStruct[4]	Structure to define State field properties
0x0120	unsigned char	bStateArea	=0. State field is located in standard input area of channel 0
	unsigned char	bStateTypeID	=5. Corresponds to a bit list (one bit per 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 periphery fault list. See description of the periphery fault list above

Table 29: Extended Status Block for AS-interface-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 within the 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 also section 2.2.1 of the netX Dual-Port-Memory manual.)

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 30: Communication Control Block

Communication Control Block Structure

```
typedef struct tagNETX_CONTROL_BLOCK
{
  UINT32 ulApplicationCOS;
  UINT32 ulDeviceWatchdog;
} NETX_CONTROL_BLOCK;
```

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 AS-Interface Master Protocol API.

4.1 Overview about Essential Functionality

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

Topic	Section Number	Section Name
initialization	6.1.1	ASI_MASTER_INITIALIZE_REQ/CNF – Initialize Master
	6.1.6	ASI_MASTER_SET_BUS_PARAM_REQ/CNF – Set Bus Parameters
	6.1.4	ASI_MASTER_SET_SLAVE_PARAM_REQ/CNF – Set Slave Parameters
Cyclic data transfer (Input/Output)	6.1.28	ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF – Set Data Exchange Active
Acyclic data transfer (Records)	6.1.23	ASI_MASTER_READ_IDENTIFICATION_STRING_REQ/CNF – Read Identification String
	6.1.24	ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ/CNF – Read Diagnostic String
	6.1.25	ASI_MASTER_READ_PARAMETER_STRING_REQ/CNF – Read Parameter String

Table 31: 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 `ASI_MASTER_SET_BUS_PARAM_REQ/CNF` packet has to be sent to the protocol stack. For more information how to accomplish this, please refer to section `ASI_MASTER_SET_BUS_PARAM_REQ/CNF` – Set Bus Parameters of this document.

Similarly, in order to configure the slaves and to supply them with warmstart parameters, the `ASI_MASTER_SET_SLAVE_PARAM_REQ/CNF` packet has to be sent to the protocol stack. More information about this topic can be obtained at section `ASI_MASTER_SET_SLAVE_PARAM_REQ/CNF` – Set Slave Parameters of this document.

4.2.2 Using the configuration tool SYCON.NET

The easiest way to configure the AS-Interface Master is using Hilscher's configuration tool SYCON.net.

- First, you need to create a project in SYCON.net. This is described in detail in the SYCON.net documentation.
- Configure the bus and master parameters as described in the SYCON.net documentation.
- After you completed your project, you can right-click on the icon of the AS-Interface Master and select "*Connect*".
- You will see that the name of the AS-Interface Master will get a green background. Now right-click on the icon again and select "*Download*".
- This will download the configuration file to the device. It is stored in a RAM Disk in a channel dependent directory ("`PORT_0`" for channel 0, "`PORT_1`" for channel 1, etc.).
- After the download is finished, the driver requests the AS-Interface Master firmware to perform a Channel-Init. All current connections will be shut down by the firmware and a restart will be performed.
- During this restart, the configuration that has been downloaded previously will be evaluated and used.

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 AS-Interface Master network.

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

Parameter	Meaning	Range of Value / Value
ulOperationMode	Operation mode.	0,1
ulDataExchange	Data exchange.	0-2
ulAutoAddress	Auto address enable.	0,1
ulProcessDataMode	Process data mode.	0,1
ulAutoClearMode	Auto-clear	0-3
ulDataFormatMode	Process data format.	0,1

Table 32: Bus and Master Parameters, their Meanings and their Ranges of allowed Values

The bus parameters to be set are in detail:

4.2.3.1 Operation Mode

This parameter describes the operation mode. It can be either protected (= 0) or configuration (=1).

Setting the operation mode to protected will cause the master to switch to protected mode. In the protected mode, the AS-Interface Master activates only projected slaves, i.e. slaves that are members of the List of Projected Slaves (LPS) and whose Configuration Data Image (CDI) and Permanent Configuration Data (PCD) are identical will only be activated. Additionally, the projected input/output configuration and all identification codes must match to the connected slave.

In configuration mode, all detected slaves will be activated by the AS-Interface Master with no respect to the differences in the values of the Configuration Data Image and Permanent Configuration Data and independently of LPS.

If an AS-Interface Slave with zero address is detected, the operation mode cannot be changed to protected mode.

Setting the operation mode can also be accomplished by sending the ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode packet to the AS-Interface Master protocol stack.

4.2.3.2 Data Exchange

This parameter describes the data exchange mode. It can be inactive, active (in online mode) or active in offline mode.

If set to active, the Master will activate the data exchange automatically after performing the reset or warm start procedure.

Setting data exchange to active state means, the AS-Interface Master cyclically sends data exchange requests to all activated Slaves. Active state is possible either online or offline.

Setting data exchange to inactive state means, the AS-Interface Master cyclically reads the identification codes from all activated Slaves, no data exchange will be performed.

Setting the data exchange to active or inactive can also be accomplished by sending the `ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF` – Set Data Exchange Active packet to the AS-Interface Master protocol stack.

4.2.3.3 Auto Addressing mode

This parameter describes the auto addressing mode. It can be enabled or disabled.

If set, the AS-Interface Master will process the function Automatic Address Assignment. This function allows replacing a defective slave by a slave with zero address. The new slave will be programmed with the address of the defective slave device, if the new slave holds the same configuration data as the slave to be replaced.

Note If the auto addressing mode is set and Auto Clear is unequal to `ASI_MASTER_AUTO_CLEAR_MODE_INACTIVE` the Master generates an initialization error.

If the master detects a slave which can replace a missing slave (it needs the same I/O Code, ID, ID1 and ID2 Code) it assigns the address of the missing slave to the detected one, if the new slave has zero address and auto addressing is enabled. Auto addressing can only be processed if one, and only one projected slave is missing.

Setting the auto addressing mode to enabled or disabled can also be accomplished by sending the `ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ/CNF` – Set Auto Address Enable packet to the AS-Interface Master protocol stack.

Checking, whether the auto addressing mode has been set to enabled or disabled can also be accomplished by sending the `ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_REQ/CNF` – Get Auto Address Enable packet to the AS-Interface Master protocol stack.

4.2.3.4 Process Data Mode

This parameter describes the process data mode. It can be set to the configured offset or a fixed offset.

If set to fixed offset, the Master will map the input and output data to an offset equal to the slaves address. So for example slave 1 of channel 0 has an input and output offset of 1, slave address 62 similarly an offset of 62. Configured offsets will be ignored.

4.2.3.5 Auto Clear Mode

This parameter describes the auto clear mode. The auto clear mode determines the error reaction behavior of the AS-Interface Master.

The following four options are available:

`ASI_MASTER_AUTO_CLEAR_MODE_INACTIVE = 0`

The Master does not take care about the status of any connected slave devices and will continue communicating with all activated slaves.

`ASI_MASTER_AUTO_CLEAR_MODE_MISSING = 1`

The Master will stop communication to the respective channel (0 or 1) if it detects at least one device as missing by changing to offline state.

ASI_MASTER_AUTO_CLEAR_MODE_FAULT = 2

The Master will stop communication to the respective channel (0 or 1) if it detects at least one device reporting a periphery fault by changing to offline state.

ASI_MASTER_AUTO_CLEAR_MODE_MISSING_FAULT = 3

The Master will stop communication to the respective channel (0 or 1) if it detects at least one device as missing or reporting a periphery fault by changing to offline state.

4.2.3.6 Data Format Mode

This parameter is used to specify which storage format to apply for word-aligned values of slave devices

- For INTEL ("LSB/MSB") data representation:

ASI_MASTER_DATA_FORMAT_MODE_BIG_ENDIAN = 0 (default)

- For MOTOROLA ("MSB/LSB") data representation:

ASI_MASTER_DATA_FORMAT_MODE_LITTLE_ENDIAN = 1

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 pages inform about the relevant slave parameters for the AS-Interface Master firmware and provide an explanation of the meaning of the parameters and ranges of allowed values.

The slave parameter structure looks as follows:

```
typedef struct ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
    ASI_MASTER_SLAVE_PARAM_HEADER_T;

struct ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
{
    /* general device description section */
    TLR_UINT16 usDataSetLen;
    TLR_UINT8  bDevFlag;
    TLR_UINT8  bParam;
    TLR_UINT8  bIoConfig;
    TLR_UINT8  bIdCode;
    TLR_UINT8  bId1Code;
    TLR_UINT8  bId2Code;

    TLR_UINT8  bVersionFlags;
    TLR_UINT8  bConfigFlags;
    TLR_UINT8  abOctet[6];
};

typedef struct ASI_MASTER_SLAVE_PARAM_Ttag
    ASI_MASTER_SLAVE_PARAM_T;

struct ASI_MASTER_SLAVE_PARAM_Ttag
{
    TLR_UINT8  bNotUsed;
    TLR_UINT8  bSlaveAddr;
    TLR_UINT8  bAction;
    TLR_UINT8  bReserved;

    ASI_MASTER_SLAVE_PARAM_HEADER_T tSlaveParamHeader;

    TLR_UINT8  abParamData[sizeof(ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T)+
        sizeof(ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T)];
};
```

Note: For details of ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T and ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T see below on page 62 and following pages.

4.2.4.1 bNotUsed

This 8-bit parameter is not used currently.

4.2.4.2 bSlaveAddr

This 8-bit parameter contains the slave address. The allowed range is 1 to 31 when standard addressing is used and 1 to 62 when extended addressing is used.

For more information regarding standard and extended addressing, see section “Slave Types and Addressing Mechanisms” of this document.

4.2.4.3 bAction

This 8-bit parameter contains the action intended to perform with the chosen AS-interface slave. The following values are possible:

Value	Code	Meaning
0x01	ASI_MASTER_ACTION_SET_SLAVE_PARAM	Set a single slave parameter
0x02	ASI_MASTER_ACTION_CLEAR_SLAVE_PARAM	Clear a single slave parameter
0x03	ASI_MASTER_ACTION_CLEAR_ALL_SLAVE_PARAM	Clear all slave parameters

Table 33: Possible Values of *bAction* and their Meanings

4.2.4.4 bReserved

This 8-bit parameter is reserved.

4.2.4.5 tSlaveParamHeader

This is a structure of type `ASI_MASTER_SLAVE_PARAM_HEADER_T`. It is structured as follows:

Slave Parameters, Meanings and Ranges

Parameter	Meaning	Range of Values
<code>usDataSetLen</code>	Length of the whole data set including this length parameter itself	
<code>bDevFlag</code>	Flag to activate the data set	
<code>bParam</code>	Slave parameter value to be stored into the Permanent Parameter (PP) area	
<code>bIoConfig</code>	IO code of the slave to be stored into the Permanent Configuration Data (PCD) area	
<code>bIdCode</code>	ID code of the slave to be stored into the Permanent Configuration Data (PCD) area	
<code>bId1Code</code>	ID1 code of the slave to be stored into the Permanent Configuration Data (PCD) area	
<code>bId2Code</code>	ID2 code of the slave to be stored into the Permanent Configuration Data (PCD) area	
<code>bVersionFlags</code>	Flag for usage of AS-Interface slaves with extended ID1 and ID2 code.	
<code>bConfigFlags</code>	Configuration flags	
<code>abOctet[6]</code>	6 byte reserved for further use	

Table 34: Slave Parameters, their Meanings and their Ranges of allowed Values

usDataSetLen

The length indicator `usDataSetLen` describes the length of the whole data block including the length of itself. The length shall be given as the length in bytes.

The following formula allows to calculate the length:

$$\text{usDataSetLen} = 16 + \text{usCfgDataLen} + \text{usOffsetDescrLen}$$

bDevFlag

The parameter `bDevFlag` declares the data set as active or inactive. Only if the active flag is set the DEVICE will activate the network access for these devices.

D7	D6	D5	D4	D3	D2	D1	D0
Active	Reserved for further use						

Table 35: Meaning of `bDevFlag`

bParam

The parameter `bParam` contains the value to be stored into the Permanent Parameter (PP) area. Valid values are in the range of 0..15 (0x0 .. 0x0F). The default value is 0.

bIoCode, bIdCode, bId1Code, bId2Code

The parameters `bIoCode`, `bIdCode`, `bId1Code` and `bId2Code` contain the input/output configuration and identification code of the slave. Valid values are in the range of 0..15 (0x0 .. 0x0F) for all parameters. Also see section „Relation between IO Config, ID Codes and Slave Profiles“ for more information.

bVersionFlags

The Parameter `bVersionFlags` is used for compatibility with earlier ASi-Interface versions.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved for further use					ID1/2	Reserved for further use	
					1	Usage of Slaves with Extended ID1 and ID2 Code	
					0	No usage of Slaves with Extended ID1 and ID2 Code	

Table 36: Meaning of `bVersionFlags`

Slaves which conform to an older version of the Complete Specification than 2.11 (see Ref. 3) do not have Extended ID codes. However, if the flag ID1/2 is set for those slaves the AS-Interface master assigns the default Extended ID codes of 0Fh accordingly.

bConfigFlags

This parameter contains configuration flags.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved for further use				ASI_MASTER_CONFIG_FLAG_RESTART	ASI_MASTER_CONFIG_FLAG_IGNORE_LPS	ASI_MASTER_CONFIG_FLAG_IGNORE_PCD	ASI_MASTER_CONFIG_FLAG_IGNORE_PP

Table 37: Meaning of `bConfigFlags`

- If `ASI_MASTER_CONFIG_FLAG_IGNORE_PP = 1`, the permanent parameter area (PP) is ignored
- If `ASI_MASTER_CONFIG_FLAG_IGNORE_PCD = 1`, the permanent configuration data (PCD) is ignored
- If `ASI_MASTER_CONFIG_FLAG_IGNORE_LPS = 1`, the list of projected slaves (LPS) is ignored
- If `ASI_MASTER_CONFIG_FLAG_IGNORE_RESTART = 1`, restarting is ignored

abOctet[6]

This parameter contains a field reserving space for future use.

4.2.4.6 abParamData[]

This array contains parameter data. It is structured into a data type description part and an offset description part. This array has a data type description part and a configuration offset part.

The data type description part looks like:

```

/*****
/** type of <code>ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_Ttag</code> */
typedef struct ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_Ttag
    ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T;

#define ASI_MASTER_SLAVE_PARAM_BIT_DATA_LEN_MSK    0x00FF
#define ASI_MASTER_SLAVE_PARAM_DIRECTION_MSK      0x8000

struct ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_Ttag
{
    TLR_UINT16 usDataTypeDescrLen;
    TLR_UINT16 ausDataType[ASI_MASTER_MAX_DESCR_COUNT];
};
/*****/

```

- **Parameter usDataTypeDescrLen**

The 16-bit parameter `usDataTypeDescrLen` contains the length of the following I/O data description inclusive the length of itself. The length shall be given as a length in bytes.

- **Array ausDataType[4]**

The array `ausDataType[4]` consisting of four 16 bit values is used in order to arrange the process data modules (input/output) for the corresponding slave device. An entry in this table has to result in a corresponding entry in the `ausIoOffset[4]` array, which contains the offset address within the dual-port memory.

Each `ausTypeData` entry is based on the following structure:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR		RFU						Process Data Length							
								Process data length in multiple of bits							
		Reserved for further use													
Data direction 0 = Module type has input data 1 = Module type has output data															

Note: Every standard slave device is treated as a 4 bit module, regardless its real number of input/output bits. A 4 bit device consumes always one octet within the dual-port memory.

For each analog slave (S-7.3 or S-7.4) only one module may be configured, because an analog slave is either an input or an output slave. For an analog slave, the configured process data length can be 16, 32 or 64 bit, because each channel requires 16 bit and an analog slave can have one two or four channels.

Example 1:

The slave is an I/O device with device profile S-3.0 (2 in/2 out) the table may look like:

Name	Value	Description
usDataTypeDescrLen	0x0006	Length in bytes, including this length indicator
ausTypeData[0]	0x0004	4 bit input process data
ausTypeData[1]	0x8004	4 bit output process data

Table 38: Example of data type description part for an I/O device with device profile S-3.0 (2 in/2 out)

Example 2:

The slave is a device with the profile S-7.3 (here: analog input module, 2 channels) the table may look like:

Name	Value	Description
usDataTypeDescrLen	0x0004	Length in bytes, including this length indicator
ausTypeData[0]	0x0020	32 bit input process data

Table 39: Example of data type description part for a device with the profile S-7.3 (here: analog input module, 2 channels)

The configuration offset part looks like:

```

/*****
/** type of <code>ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_Ttag</code> */
typedef struct ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_Ttag
    ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T;

#define ASI_MASTER_MAX_DESCR_COUNT                4

struct ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_Ttag
{
    TLR_UINT16 usOffsetDescrLen;
    TLR_UINT8  bInputCnt;
    TLR_UINT8  bOutputCnt;
    TLR_UINT16 ausOffset[ASI_MASTER_MAX_DESCR_COUNT];
};
/*****

```

- **Parameter usOffsetDescrLen**

The 16-bit parameter `usOffsetDescrLen` contains the length of the array `ausOffset[4]` including the length of itself. The length is to be specified in units of bytes.

- **Parameter bInputCnt**

This 8-bit parameter determines the number of input description entries to be applied.

- **Parameter bOutputCnt**

This 8-bit parameter determines the number of output description entries to be applied.

- **Parameter `ausOffset[4]`**

The array `ausOffset[4]` is used in order to specify the offset address for an input or output module of a slave device. An entry in this table has to result in a corresponding entry in the `ausTypeData[...]` array, which contains the length and the direction of the module.

If the AS-Interface slave device consists of input and output modules, the first entry has to be the description for the input module. All offsets has to be configured as byte offsets, because each slave device, except an analogue device, is treated as a 4 bit module, regardless its real number of input/output bits.

Each `ausOffset` entry is based on the following structure:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RFU							Byte offset								
Reserved for further use							Byte offset within the dual-port memory								

Example 1:

The slave is an I/O device with the profile S-3.0 (2 in / 2 out). The output module has to be mapped to the byte offset 4 and the input module has to be mapped to byte offset 2, the table has to look like:

Name	Value	Description
<code>usOffsetDescrLen</code>	0x0008	Length in byte, inclusive this length indicator
<code>bInputCnt</code>	0x0001	1 input offset description following
<code>bOutputCnt</code>	0x0001	1 output offset description following
<code>ausOffset[0]</code>	0x0002	Input module mapped to byte offset 2
<code>ausOffset[1]</code>	0x0004	Output module mapped to byte offset 4

Table 40: Example of configuration offset part for an I/O device with device profile S-3.0 (2 in/2 out)

Example 2:

The slave is a device with the profile S-7.3 (here: analog input module). The input module has to be mapped to the byte offset 5, the table has to look like:

Name	Value	Description
<code>usOffsetDescrLen</code>	0x0006	Length in byte, inclusive this length indicator
<code>bInputCnt</code>	0x0001	1 input offset descriptions is following
<code>bOutputCnt</code>	0x0000	No output offset description following
<code>ausOffset[0]</code>	0x0005	Input module mapped to offset 5

Table 41: Example of configuration offset part for a device with the profile S-7.3 (here: analog input module, 2 channels)

4.3 Task Structure of the AS-Interface Master Stack

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

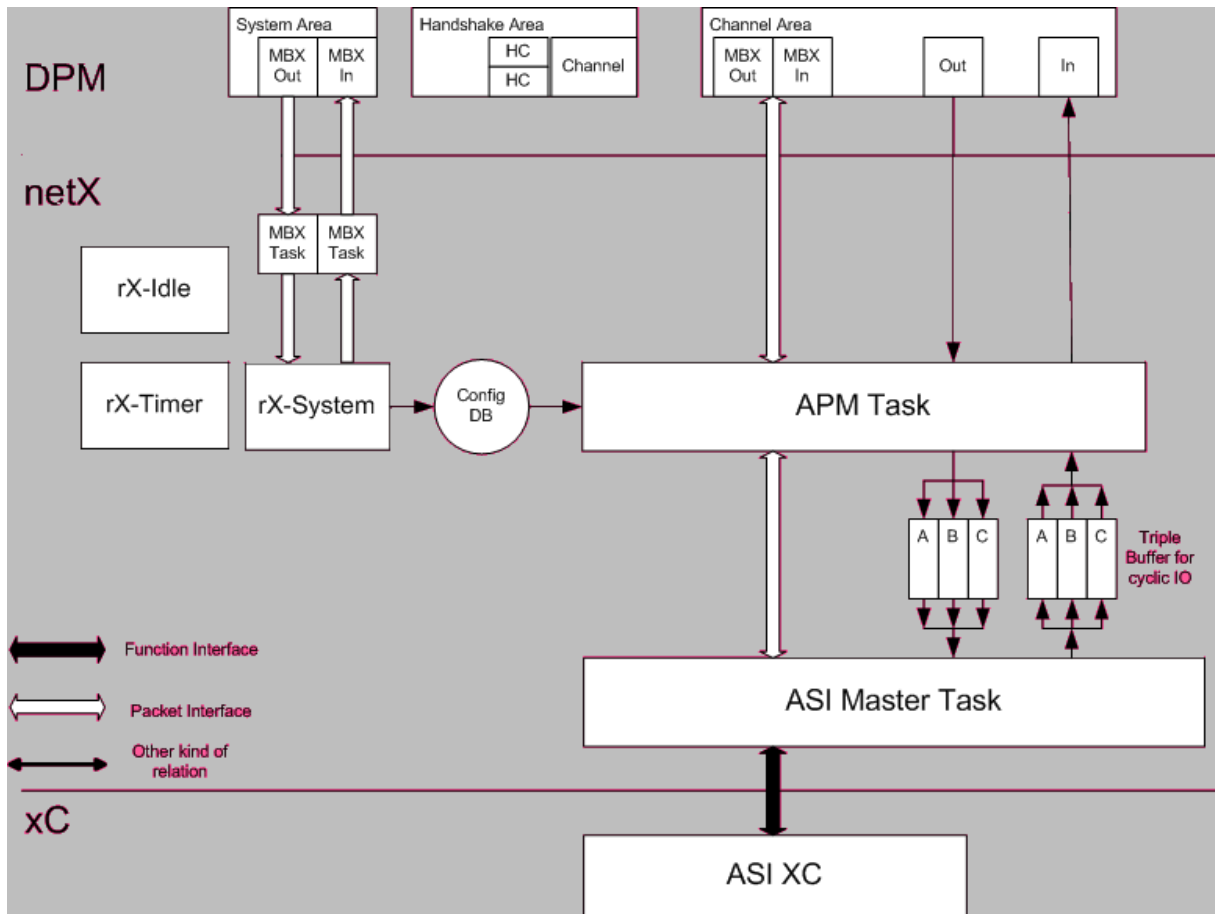


Figure 6: Task Structure of the AS-Interface Master Stack:

The tasks of the AS-Interface Master stack are used for the following purposes:

- The ASIMASTER task represents the core of the AS-Interface Master stack.
- The AP task represents the interface between the AS-Interface Master protocol stack and the dual-port memory. It is responsible for:
 - Control of LEDs
 - Diagnosis
 - Packet routing
 - Update of the IO data

The triple buffer mechanism provides a consistent synchronous access procedure from both sides (DPM and AP task). The triple buffer technique ensures that the access will always affect the last written cell.

5 Overview

5.1 Introduction to AS Interface

AS Interface (abbreviation for *Actuator-Sensor Interface*) is a networking system for connecting binary and analog sensors and actuators with control equipment. It is located at the lowest Field Bus area.

It is especially suited for applications requiring for a simple protocol sequence and a high degree of operational safety in a rough industrial environment.

The AS Interface system is designed as a Single-Master-System, which means that there is always only one AS Interface Master per line and this controls the data exchange. Using the standard addressing method up to 31 AS Interface Slaves can be connected to the AS Interface Master. However, if extended addressing is used, connecting up to 62 AS Interface Slaves is possible.

The allowed address range extends

- from 1 to 31 in case of standard addressing.
- from 1A/1B to 31A/31B in case of extended addressing.

Each AS Interface slave has to store its address in a non-volatile memory in order to allow later address changes. Only slaves with a currently valid address may participate in the network, i.e. they may answer to data and parameter requests from the AS Interface Master.

If an address is duplicated, then the A or B address is called up in each ASi cycle. In this way, the time in which the data are updated with the Slave doubles to approximately 10 ms. The transmission of an analog value takes at least 8 cycles and is not limited in time.

The bus access method applied within the AS-Interface system is a master-slave method with cyclic polling. This means in detail, that the AS Interface Master calls up each AS Interface Slave in one-by-one sequence and awaits its response. It is ensured that within a cycle time of maximum 5 ms, a data telegram can be exchanged with all 31 AS Interface Standard Slaves. If extended addressing is applied, 62 Extended AS Interface Slaves can be serviced within 10 ms.

Functions for the parameterizing, configuration, automatic address assignment and monitoring inside the network are carried out a-cyclically.

AS Interface makes use of constant message lengths. Complex procedures for sending control and characterization of message lengths and data formats are not required.

5.1.1 Data Convention

The following convention applies to data in AS Interface:

Viewed from the host, the value 0 indicates that a sensor, actuator or another slave function is switched off or inactive, while the value 1 indicates that this sensor, actuator or slave function is switched on or active.

However, there is one exception: This general rule does not apply on data output. In this special case the data are inverted (by the master), so that 0 means active and 1 means inactive.

You should also have in mind that only data output, but not parameter output is affected by this exception!

5.2 AS-Interface Master

This section describes the features and characteristics of the AS-Interface Master.

The following topics are covered:

5.2.1 General Structure of the AS-Interface Master

In general, the AS-Interface Master consists of these three parts:

- Host interface
- Execution Control
- Transmission Control

The host interface acts as an interface between the master and the host. It therefore provides functionality for the communication between the host and the master. The host interface of all Hilscher netX-based communication products with AS-Interface Master functionality is a packet interface.

For a more detailed description, see section *The Application Interface* of this document.

Execution control has the following tasks:

- Initialization of both the master itself and the complete network.
- Cyclic data exchange
- Acyclic command execution
- Providing various execution control functions

Additionally, execution control provides procedures both for initialization and normal operation and also for administrative purposes.

Finally, execution control provides a lot of lists for administrative purposes, see section *Lists maintained by Execution Control* on page 73 of this document.

Transmission control provides the infrastructure for sending a single request from the master to the slave via the AS-Interface network.

Furthermore, transmission control is responsible for the detection of transmission errors such as>

- Missing slave responses
- Incorrect slave responses

Additionally, transmission control provides transactional support for two kinds of transactions, namely single transactions and combined transactions. These are described in a separate section of this chapter, see section *Transmission Control* on page 77 and the following for more information.

5.2.2 Execution Control

5.2.2.1 Introduction

Execution Control is based on Transaction Control. It provides procedures for two main purposes:

- Procedures for start-up of the AS-Interface Master and its normal operation
- Administrative procedures, invoked by the host interface

Automatic address assignment to slaves is also performed by execution control.

In the start-up operation, the AS Interface Master attempts to acquire and activate all the functional Slaves in the network.

Here, a difference is made between two different modes of operation:

- Protected mode: Only configured Slaves are activated.
- Configuration mode: All Slaves are activated.

More information about these operation modes is available at section of this document.

There are 6 different phases of AS-Interface Master operation:

The following three phases apply to start-up operation:

- Offline Phase
- Detection Phase
- Activation Phase

The following three phases apply to normal operation:

- Data Exchange Phase
- Management Phase
- Inclusion Phase

More information about these phases of operation is available at section *Operation Modes* of this document.

Additionally, execution control maintains a couple of lists for administrative purposes. These lists can be ordered in four groups:

- Input/ output data images in the master (IDI, ODI, AIDI, AODI)
- Configuration data images in the master (CDI, PCD)
- Parameter images in the master (PI, PP)
- Lists of slaves in the master (LPS, LAS, LDS, LPF)

For more information, see section *Lists maintained by Execution Control*.

Finally, execution control provides 30 administrative functions.

5.2.2.2 Operation Modes

There are two possible operation modes in AS-Interface. These are:

- Configuration mode
- Protected operating mode

Configuration mode

This type of operation permits start up of the system without previous configuration. All recognized Slaves are included in the data exchange.

Protected operating mode

In this operating mode, the AS Interface Master communicates only with the planned AS-Interface Slaves. Missing or incorrectly connected slaves will cause an error message.

5.2.2.3 Transmission Phases

The following transmission phases can occur at an AS-Interface Master device:

Offline Phase

Offline phase occurs immediately after power-up of the AS-Interface Master. During offline phase, the internal lists needed for operation are initialized. At the end of offline phase, all internal preparations for operation within the master have been performed and the master is ready to communicate with slaves.

Detection Phase

Detection Phase is the phase following next to offline phase. The AS-Interface Master tries detecting slaves. In order to accomplish this, it scans its entire address range for detecting slaves. At the end of detection phase, the AS-Interface Master is aware at which slave addresses of its network. AS-Interface Slaves are present, and at which not.

Activation Phase

Activation phase is the phase following next to detection phase. It is used to activate the AS-Interface Slaves. The condition for a slave to be activated within activation phase depends on the current mode of operation:

- Protected mode: Only these detected AS-Interface Slaves which have been pre-configured are activated. This also requires the detected configuration and identification codes to match the configured values.
- Configuration mode: All detected AS-Interface Slaves are activated.

At the end of offline phase, the preparations for normal operation have been completed and thus normal operation will take place as starting up has finished. This especially means switching to data exchange phase for the first time since power-up. From now on, the AS-Interface Master will be in one of the following three states of normal operation.

Data Exchange Phase

The actual useful data exchange takes place in the cyclic normal operation. A cycle consists sequentially of data exchange, management and acceptance phase.

In the data exchange phase, the Master sequentially contacts all the activated Slaves with a call for data. This call-up is called polling and it includes a depiction of the output data for the corresponding Slave. The called AS-Interface Slave will answer by transferring its input data.

This means, the AS-Interface Master polls all slaves having been activated in the activation phase. Each slave is polled once within the data exchange phase. This happens in a fixed order. At the end of the data exchange phase, all activated AS-Interface Slaves have been polled and one cycle of cyclic communication has thus been finished. One current slave data image has been collected within the AS-Interface Master. Now is the time for performing acyclic and administrative operations before the next cycle begins. To do so, the AS-Interface Master will switch either to management phase or to inclusion phase..

Management Phase

As stated above, this phase may be started at the end of data exchange phase in order to perform acyclic operations. This means data are acyclically sent once to exactly one single slave.

In the management phase, tasks can be carried out at the Slaves. These tasks are transmitted "a-cyclically" and, under certain circumstances, are spread over several management phases. If there is no task, the acceptance phase is immediately proceeded with. Tasks that can be carried out during the management phase are:

- Data exchange with a single Slave,
- Setting the parameter value of a Slave,
- Setting the extended ID1 Code of Slave 0,
- Reading the status of a Slave,
- Reading the I/O configuration of a Slave,
- Reading the ID Code of a Slaves,
- Reading the extended ID1 Code of a Slave,
- Reading the extended ID2 Code of a Slave,
- Deleting the address of a Slave,
- Setting the address of a Slave to 0,
- Allocating a new address to Slave 0,
- Resetting a Slave,
- Resetting all Slaves (Broadcast Reset),
- Performing a R1 command.

At the end of management phase, the AS-Interface Master will switch to inclusion phase.

Inclusion Phase

Finally, the AS-Interface Master will have to check for changes within its network. For instance it will check for new AS-Interface Slaves which might have been added to the network in the mean time. This is done by sending calls to free addresses which have not been occupied by slaves yet. If an answer is received, the slave can be included into the AS-Interface interface network. Also the status of all connected Slaves is queried within the inclusion phase (for instance, the information of the periphery fault list is collected in the inclusion phase).

At the end of inclusion phase, the AS-Interface Master will again switch to data exchange phase in order to start a new communication cycle .

5.2.2.4 Execution Control Functions

Execution control may provide up to 30 administrative functions which are listed in the complete specification (see reference # 3). The following is a list of functions which are provided by the packet interface of the AS-Interface Master Protocol Stack and the associated packets.

Function #	Function Name (within AS-Interface Complete Specification)	Packet in AS-Interface Master protocol Stack
4	Get_Permanent_Parameter	ASI_MASTER_GET_PERMANENT_PARAMETER_REQ/CNF
5	Write_Parameter	ASI_MASTER_WRITE_PARAMETER_REQ/CNF
9	Get_Permanent_Configuration	ASI_MASTER_GET_PERMANENT_CONFIG_REQ /CNF
11	Read_Actual_Configuration	ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF
13	Get_LPS	ASI_MASTER_GET_LPS_REQ /CNF
14	Get_LAS	ASI_MASTER_GET_LAS_REQ/CNF
15	Get_LDS	ASI_MASTER_GET_LDS_REQ/CNF
16	Get_Flags	ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_REQ/CNF
17	Set_Operation_Mode	ASI_MASTER_SET_OPERATION_MODE_REQ/CNF
18	Set_Offline_Mode	ASI_MASTER_SET_OFFLINE_MODE_REQ/CNF
19	Set_Data_Exchange_Active	ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF
20	Change_Slave_Address	ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ/CNF
21.1	Set_Auto_Address_Enable	ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ/CNF
21.2	Get_Auto_Address_Enable	ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_REQ/CNF
22	Execute_Command	ASI_MASTER_EXECUTE_COMMAND_REQ/CNF
23	Get_LPF	ASI_MASTER_GET_LPF_REQ/CNF
24	Write_Extended_ID_1	ASI_MASTER_WRITE_EXT_ID1_CODE_REQ/CNF
27	Read_Parameter_String	ASI_MASTER_READ_PARAMETER_STRING_REQ/CNF
28	Write_Parameter_String	ASI_MASTER_WRITE_PARAMETER_STRING_REQ/CNF
29	Read_Diagnostic_String	ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ/CNF
30	Read_Identification_String	ASI_MASTER_READ_IDENTIFICATION_STRING_REQ/CNF

Table 42: Relation between Execution Control Functions and the corresponding Packets

Functions

1 (Read_IDI),
2 (Write_ODI),
3 (Set_Permanent_Parameter),
6 (Read_Parameter),
7 (Store_Actual_Parameters),
8 (Set_Permanent_Configuration),
10 (Store_Actual_Configuration),
12 (Set_LPS),
25 (Read_AIDI) and
26 (Write_AODI) defined in the complete specification of AS-Interface currently do not have a corresponding packet within the packet interface of the AS-Interface Master Protocol Stack. The functionality of functions 1, 2, 25 and 26 can also be achieved using the ASI_MASTER_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle packet.

5.2.3 Lists maintained by Execution Control

This section describes the lists maintained by execution control. Execution control maintains these lists in order to perform communication between the AS-Interface Master and the AS-Interface Slaves and to offer diagnostic possibilities.

These lists can be ordered in four categories:

- Input/ output data images in the master (IDI, ODI, AIDI, AODI)
These contain data read from the slaves or to be written to the AS-Interface Slaves.
- Configuration data images in the master (CDI, PCD)
These contain configuration data and identification codes of the AS-Interface Slaves.
- Parameter images in the master (PI, PP)
These contain parameter data read from the slaves or to be written to the AS-Interface Slaves.
- Lists of slaves in the master (LPS, LAS, LDS, LPF)
These contain lists of configured, active and detected slaves and of slaves with peripheral faults.

They are described in the subsequent sections.

5.2.3.1 Input Data Image (IDI)

The input data image is the process data image for input, i.e. the memory area into which incoming data are transferred. You can access the IDI by retrieving a handle using the packet `ASI_MASTER_GET_BUFFER_HANDLE_REQ/CNF` – Get Buffer Handle (Handle to receive buffer) and reading out this memory address.

The size of the IDI is 32 bytes.

The first 16 bytes are used for storage of the bits of the standard AS-Interface Slaves and the A-Slaves. The second 16 bytes are exclusively used for the storage of the data from B-Slaves. For more information on extended addressing with A- and B-Slaves see section *Slaves with extended Address Regions* on page 81 of this document.

The contents of the input data image is structured as follows:

Let n be the starting address of the IDI. The input data image begins with a reserved area of 4 bits at address n , bits 7-4. At address n , bits 3 to 0 the 4 bits of slave 1 are stored. At address $n+1$, the data of slave 2 (bits 7-4) and of slave 3 (bits 3-0) are stored. Subsequently, each further address up to $n+15$ stores the data of two more slaves in ascending order, so at address $n+15$ you can find the data of slave 30 (bits 7-4) and 31 (bits 3-0).

The IDI is initialized by setting it to the default value (0, i.e. all bits are set to the value 0).

5.2.3.2 Output Data Image (ODI)

The output data image is the process data image for output, i.e. the memory area into which data to be sent are transferred. You can access the ODI by retrieving a handle using the packet `ASI_MASTER_GET_BUFFER_HANDLE_REQ/CNF` – Get Buffer Handle (Handle to send buffer) and reading out this memory address.

The size of the ODI is 32 bytes.

The first 16 bytes are used for storage of the bits of the standard AS-Interface Slaves and the A-Slaves. The second 16 bytes are exclusively used for the storage of the data from B-Slaves. For more information on extended addressing with A- and B-Slaves see section *Slaves with extended Address Regions* on page 81 of this document.

The contents of the output data image is structured similar to that of the IDI.

The ODI is initialized by setting it to the default value (0xF, i.e. all bits are set to the value 1).

5.2.3.3 Analog Input Data Image (AIDI)

Similar as IDI, but used for storage of analog input data. The length can be up to 256 bytes. Data are stored in ascending order organized as 16 bit words.

5.2.3.4 Analog Input Data Image (AODI)

Similar as ODI, but used for storage of analog output data. The length can be up to 256 bytes. Data are stored in ascending order organized as 16 bit words.

5.2.3.5 Configuration Data Image (CDI)

The configuration data image contains

- the current IO-Configuration,
- the ID-Code,
- the Extended ID1-Code
- and the Extended ID2-Code,

of all AS-Interface Slaves, determined by reading out these data from the AS-Interface Slaves.

The configuration data image of inactive AS-Interface Slaves is initialized by setting it to the default value (0xF, i.e. all bits are set to the value 1).

Accessing the Configuration Data Image is accomplished by packet `ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF` – Read Configuration, see description on page 126.

5.2.3.6 Permanent Configuration Data (PCD)

The permanent configuration data contains the configured parameter output, i.e.

- the current IO-Configuration,
- the ID-Code,
- the Extended ID1-Code
- and the Extended ID2-Code,

of all AS-Interface Slaves, determined by the local configuration of the `ASIMASTER` Task.

The permanent configuration data of AS-Interface Slaves which have not been projected is initialized by setting it to the default value (0xF, i.e. all bits are set to the value 1).

Accessing the Permanent Configuration Data is accomplished by packet `ASI_MASTER_GET_PERMANENT_CONFIG_REQ/CNF` – Get Permanent Configuration, see description on page 135.

5.2.3.7 Parameter Image (PI)

This array contains current copies of the parameter output of all active slaves determined by the requests of the last `Write_Parameter` function call (Execution control function 5) to the slaves.

The initialization of the parameter image is done as follows: After master power on, the execution control copies all values of the permanent parameter from non-volatile memory to the parameter image array. All other values of the PI remain unchanged.

Accessing the parameter image is accomplished by `ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF` – Read Configuration, see description on page 126.

5.2.3.8 Permanent Parameter (PP)

This array contains the configured parameters of all slaves, determined by local configuration of the master device.

Permanent parameter data are stored in non-volatile memory. After master power on, the execution control copies all values of the permanent parameter from non-volatile memory to the parameter image array, see just above.

Accessing the Permanent Parameter Area is accomplished by packet `ASI_MASTER_GET_PERMANENT_PARAMETER_REQ/CNF` – Get Permanent Parameter, see description on page 139.

5.2.3.9 List of Projected Slaves (LPS)

This list contains one bit for each slave that has been configured.

Accessing the List of Projected Slaves is accomplished by packet `ASI_MASTER_GET_LPS_REQ/CNF` – Get List of Projected Slaves, see description on page 161.

5.2.3.10 List of Activated Slaves (LAS)

This list contains one bit for each slave that has been activated and actually detected by the master during start up or inclusion phase.

The List of Activated Slaves is initialized by setting it to the default value (0, i.e. all bits are set to the value 0).

Accessing the List of Activated Slaves is accomplished by packet `ASI_MASTER_GET_LAS_REQ/CNF` – Get List of Activated Slaves, see description on page 164.

5.2.3.11 List of Detected Slaves (LDS)

This list contains one bit for each slave that is detected by the master during start up or inclusion phase.

The List of Detected Slaves is initialized by setting it to the default value (0, i.e. all bits are set to the value 0).

Accessing the List of Detected Slaves is accomplished by packet `ASI_MASTER_GET_LDS_REQ/CNF` – Get List of Detected Slaves, see description on page 167.

5.2.3.12 List of Peripheral Faults (LPF)

This list contains all activated slaves with peripheral fault bit set to the value “1”.

The List of Peripheral Faults is initialized by setting it to the default value (0, i.e. all bits are set to the value 0).

Accessing the List of Peripheral Faults is accomplished by packet `ASI_MASTER_GET_LPF_REQ/CNF` – Get List of Peripheral Faults, see description on page 170.

5.2.4 Transmission Control

Transmission Control provides a means to send one single request from the AS-interface Master to the AS-interface Slave and to receive an answer from the slave at the master.

There is one major rule:

At one time only one single request can be processed. No other request will be processed before the processing of the preceding request has completely been finished.

This kind of request processing is also denominated as transaction processing. In case of an error one attempt for a repeated transmission can be made, more retry attempts are not allowed.

5.2.4.1 Error Handling

Transmission control is responsible for the detection of transmission errors such as:

- Missing slave responses
- Incorrect slave responses

Possible causes for missing slave responses may be:

1. No physical connection of AS-Interface Slave present.
2. No valid slave address present.
3. The slave detected a failure in the received request from the master.
4. A defective receiver prevents the slave from receiving the masters request telegrams.
5. A reset occurred at the slave.
6. Slave cannot answer anymore due to defect.
7. Slave has not been parameterized.

All these kinds of problems will cause the allowed response time of the timer to be exceeded.

Possible causes for incorrect slave responses may be:

1. Simultaneous transmission by several slaves
2. Disturbances on the line

5.2.4.2 Single Transactions

In general, there are two transaction types available in AS-Interface:

- single transactions which transfer at most 4 bits of input and output and
- transactions consisting of multiple commands which are usually denominated as combined transactions.

Single transactions of the following types are defined within AS-Interface:

Single transaction type	Action
Data_Exchange	A bit pattern is read or written from/to the data output/input of the slave
Write_Parameter	A bit pattern is read or written from/to the data output/input of the slave
Address_Assignment	Assignment of an address to be stored in the non-volatile memory of an AS-Interface Slave currently having the (intermediate) address 0. The range of allowed address depends on whether the standard addressing (1-31) or extended addressing mechanism (1A-31A,1B-31B) is used.
Commands	Commands for miscellaneous function such as reset or read-out of configuration and status data.

Table 43: Types of Single Transactions in AS-Interface

5.2.4.3 Combined Transactions (Multiple Transactions)

Combined transactions are such transactions which transfer more than 4 bit of data and therefore have to use multiple commands (single transactions). In this situation an additional set of rules is required for controlling the transfer of data. These sets of data are denominated as combined transaction types.

These combined transaction types are connected to some slave profiles and, if necessary, also master profiles.

There are 6 combined transaction types (CTT) available (CTT1 to CTT5 and CTT for Safety at work). The following table shows which slave and master profiles are associated with which CTT and what is the main application for the CTT:

CTT	Description	Slave Profile	Master Profile
Type 1	Not recommended	S-7.1	-
Type 1	Not recommended	S-7.2	-
Type 1	16 bit input or output	S-7.3	M3
Type 1	Complex field devices	S-7.4	M3
Type 2	Combi field devices	S-7.5.5	M4
Type 2	Combi field devices	S-7.A.5	M4
Type 2	Serial communication field devices	S-B.A.5	M4
Type 3	4 inputs/4 outputs in extended addressing mode	S-7.A.7	M4
Type 3	8 inputs/8 outputs in extended addressing mode	S-7.A.A	M4
Type 4	16 inputs in extended addressing mode	S-7.A.8	M4
Type 4	Dual 16 inputs in extended addressing mode	S-7.A.9	M4
Type 5	High speed 16 bit inputs and outputs	S-6.0	M4
Safety at Work	Safety at Work input slaves	S-0.B	Any profile
Safety at Work	Safety at Work input slaves with standard outputs	S-7.B	Any profile

Table 44: Available Combined Transaction Types

5.2.4.4 States of AS-Interface Master Protocol Stack

An AS-Interface Master can be in one of four different states, which are called the master states and differ by a different degree of allowed functionality. The master states are stored within the Extended Status, see page 41 and following ones.

These master states and their symbolic names are:

- OFFLINE (ASI_MASTER_MASTER_STATE_OFFLINE)
- STOP (ASI_MASTER_MASTER_STATE_STOP)
- CLEAR (ASI_MASTER_MASTER_STATE_CLEAR)
- OPERATE (ASI_MASTER_MASTER_STATE_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. 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 AS-Interface Slaves. 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 state are supervised by an internal state machine within the AS-Interface Master.

A change of the mode is indicated to the AP-task by the indication "ASI_MASTER_STATE_CHANGE_IND/RES – State Change Indication", see page 210.

5.3 AS-Interface Slave -Structure and Addressing

This section describes the features and characteristics of the AS-Interface Slave. A slave can be supervised by a watchdog timer in order to check its availability.

The following topics are covered in this section:

5.3.1 Slave Types and Addressing Mechanisms

5.3.1.1 Standard Slaves

Up to 4 input and up to 4 output data bits are available per Slave. A maximum of 31 Standard Slaves can be connected to an AS Interface. This means, a maximum of 124 purely binary sensors and 124 purely binary actuators can be connected to the AS Interface network.

5.3.1.2 Slaves with extended Address Regions

Only AS Interface Slaves with ID-Code "A" can be operated in the extended address region.

For AS Interface Slaves with the extended address region there are available up to 4 input and up to 3 output data bits. Up to a maximum of 62 Slaves with extended address regions can be connected to a network. This means, a maximum of 248 purely binary sensors and 186 purely binary actuators in the extended address region can be connected to the network.

Analog Slaves

Analog Slaves are devices with IO configuration "7" and ID Codes "1" to "4". However, according to the AS interface specification V2.11, the Master can only deal with Slaves with the ID Codes "3" or "4" as analog Slaves. Analog Slaves with ID Codes "1" or "2" are dealt with as standard Slaves and must be controlled by the user program.

Analog Slaves are offered by the manufacturer in various models. A maximum of 31 analog Slaves can be connected to an AS Interface network. Each Slave has available to it up to 4 analog input words (each 16 Bit) **or** up to 4 output words (each 16 Bit).

Structure of an AS Interface network in the extended address region

For AS-Interface Slaves with extended address region (ID Code "A"), a doubling of the addresses from 31 to 62 is possible. This occurs in that the addresses 1 to 31 are issued twice so that, for instance, address 19 is divided into the addresses 19A and 19B. Doubling the addresses for AS-Interface Slaves that are not equipped with ID Code "A" is not possible.

Thus, a maximum number of 62 Slaves is only possible when all AS-Interface Slaves in the extended address region are working. For each Standard or Analog Slave, the maximum number of connected Slaves is reduced by one. If 31 Standard or Analog slaves are connected to the network, then no doubling is possible for any address and the maximum number of connected AS-Interface Slaves is 31.

5.3.1.3 Extended Addressing Mode

For AS-Interface Slaves in extended addressing mode (ID-Code = 'A'), a doubling of the operating addresses is possible. This occurs in that the addresses from 1 to 31 are issued twice so that, for instance, address 19 is divided into the addresses 19A and 19B. Doubling the addresses for Slaves that do not have the ID-Code 'A' is not possible.

Thus, a maximum number of 62 AS-Interface Slaves is only possible when all AS-Interface Slaves are working in extended addressing mode. For each other AS-Interface Slave (ID-Code ≠ 'A'), the maximum number of connectable AS-Interface Slaves is reduced by one. If 31 AS-Interface Slaves with ID-Code unequal 'A' are connected to the network, no doubling of any address is possible and the maximum number of connectable AS-Interface Slaves is reduced to 31.

The following restrictions apply to all AS-Interface Slaves in extended addressing mode:

- Each AS-Interface Slave in extended addressing mode has only 3 parameter bits (P0 - P2), the fourth parameter bit (P3) is unused.
- Each AS-Interface Slave in extended addressing mode and not using any combined transaction type can have up to 3 bits of output data (D0 - D2), the fourth data bit (D3) is unused. The maximum input data of 4 bits is unchanged.
- Addresses can only be doubled, if **both** AS-Interface Slaves have ID-Code 'A'.

The following table defines the mapping of standard and extended addresses in the application interface.

Standard Slave address	Extended Slave address	Address in application interface
1	1A	1
2	2A	2
3	3A	3
4	4A	4
...
30	30A	30
31	31A	31
-	1B	32
-	2B	33
-	3B	34
-	4B	35
...
-	30B	61
-	31B	62

Table 45: AS-Interface Slave Addresses in the Application Interface

5.3.2 Slave Registers and Flags

There are four registers available at the AS-interface Slave:

- The Address Register
- The I/O Configuration Register
- The Data Output Register
- The Parameter Output Register

5.3.2.1 The Address Register

The Address Register is 5 bit wide.

It contains the current address of the slave. After a reset, the address information stored in non-volatile ram is copied from there to this register. However, for later address changes, the current address can be changed via the master by sending a `ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ/CNF` – Change Slave Address packet to the protocol stack, see page 155 of this document.

5.3.2.2 The I/O Code Register

The I/O Code Register is 4 bit wide.

It contains the I/O code of the four data ports. These ports are once programmed (during production of the slave) to be

- inputs
- outputs
- tristate
- or bidirectional ports

Afterwards they can only be read out, but it is not possible to change them any more.

5.3.2.3 ID Code register

The ID Code Register is 4 bit wide.

5.3.2.4 Extended ID Code register

The Extended ID Code Register is 4 bit wide.

The Extended ID-Code is handled in the same way as the 4 bit standard ID-Code.

5.3.2.5 The Data Output Register

The Data Output Register is 4 bit wide.

It contains the data of the last data request from the master having been received free of any errors at the slave. Bytes configured as output are transferred to the corresponding data port.

5.3.2.6 The Data Input Register

The Data Input Register is 4 bit wide. This register is only relevant for the 'Synchronous Data I/O Mode' of AS-Interface, which however is not supported by the current implementation.

5.3.2.7 The Parameter Output Register

The Parameter Output Register is 4 bit wide.

It contains the data of the last parameter request from the master having been received free of any errors at the slave. The bits are sent at the parameter ports of the slave. A parameter request can only be issued once in a cycle per slave.

5.3.2.8 The Receive Register

The Receive Register contains the most recently received master request (without start bit and end bit).

5.3.2.9 The Transmit Register

The Transmit Register contains the response of the AS-Interface Slave before it is transmitted to the AS-Interface Master.

5.3.2.10 The Status Register

The Status Register is 4 bit wide. These four bits are independent from each other.

It contains status information according to the following table:

Status Register Bit	Meaning	
S0	Address/Extended ID-Code 1 is stored in a volatile or non-volatile memory:	
	0: Permanent storage	1: Volatile storage
S1	A periphery fault condition occurred	
	0: no periphery fault has occurred	1: periphery fault has occurred
S2	Undefined	
	0: Undefined	1: Undefined
S3	An error occurred while reading non-volatile memory	
	0: no error occurred	1: An error occurred

Table 46: The Status Register of the AS-Interface Slave

5.3.3 Relation between IO Config, ID Codes and Slave Profiles

The AS-Interface Master can retrieve a lot of information about an AS-Interface Slave from its profile if IO Configuration and ID Codes are known as there is a fixed relation between the combination of IO-Configuration and the ID-Codes on one hand and the slave profile on the other hand allowing to determine the profile. For instance, the following characteristics of the AS-Interface Slave may depend on the profile associated with the slave:

- Contents and meaning of data bits
- Usage of parameters and their meaning if any are used
- Addressing mode
- Permission whether the periphery fault bit may be evaluated or not
- Behavior of the AS-Interface Slave
- Definition of logic signal levels at the inputs and outputs of the slave
- Time delay of input or output signals
- Pin assignments of the module

The relation between AS-Interface Slave profile and IO-Configuration and the ID-Codes is depicted in Table 47 in case of standard slaves.

Slave Profiles					ID Code															
IO Configuration					0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	I	I	I	I	0.0	0.1									0.A	0.B				0.F
1	I	I	I	O	1.0	1.1									1.A					1.F
2	I	I	I	B	2.0										R					2.F
3	I	I	O	O	3.0	3.1									3.A					3.F
4	I	I	B	B	4.0										4:A					4:F
5	I	O	O	O	5.0										5.A					5.F
6	B	B	B	B	6.0										6.A					6.F
7	B	B	B	B	7.0	7.1	7.2	7.3	7.4	7.5					7.A	7.B		7.D	7.E	7.F
8	O	O	O	O	8.0	8.1									8.A					8.F
9	O	O	O	I	R										9.A					9.F
A	O	O	O	B	A.0										R					A.F
B	O	O	I	I	R	B.1									B.A					B.F
C	O	O	B	B	C.0										C.A					C.F
D	O	I	I	I	R	D.1									D.A					D.F
E	O	B	B	B	E.0										E.A					E.F
F	T	T	T	T	For future use															

Table 47: Relation between Slave Profile and IO-Configuration and the ID-Codes for Standard Slaves

Similarly, Table 48 shows this relation in case of slaves in extended addressing modes:

Slave Profiles					Extended ID Code 2															
IO Configuration					0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	I	I	I	I	0A0		0A2												0AE	R
1	I	I	I	O	1A0														1AE	R
2	I	I	I	B	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
3	I	I	O	O	3A0	3A1	3A2												3AE	R
4	I	I	B	B	4A0														4AE	R
5	I	O	O	O	5A0														5AE	R
6	B	B	B	B	6A0														6AE	R
7	B	B	B	B	7A0		7A2		7A5		7A7	7A8	7A9	7AA					7AE	R
8	O	O	O	O	8A0		8A2												8AE	R
9	O	O	O	I	9A0														9AE	R
A	O	O	O	B	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
B	O	O	I	I	BA0		BA2		BA5										BAE	R
C	O	O	B	B	CA0														CAE	R
D	O	I	I	I	DA0														DAE	R
E	O	B	B	B	EA0														EAE	R
F	T	T	T	T	For future use															

Table 48: Relation between Slave Profile and IO-Configuration and the ID-Codes for Slaves supporting Extended Addressing according to AS-interface Specification 2.1

These tables are also described in Annex A of the Complete AS-Interface Specification), see reference 4.

The meaning of the letters within both tables is as follows:

Letter	Meaning
I	Input
O	Output
B	Bidirectional
T	Tristate
R	Reserved

Table 49: Explanation of Abbreviations in Table 47 and Table 48

For practically retrieving the desired slave information from the profile, proceed as follows:

2. The IO-Code and ID Codes (i.e. ID Code, Extended ID Code 1 and 2) can be retrieved using the ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF – Read Configuration packet described on page 126 of this document. Use the correct slave address for the slave whose profile information is to be retrieved. Using the results of this packet (returned in the confirmation packet) and the tables, you can determine the profile belonging to an AS-Interface Slave.
 3. Now you need to know whether the slave is a standard slave or can be accessed via extended addressing. This can easily be determined by evaluating the ID Code. If its hexadecimal value is equal to 0xA (i.e. binary 1010), then it is a slave supporting extended addressing according to AS-Interface specification 2.1 or higher, otherwise it is a standard slave. This information is required to choose the correct table for profile determination depending on IO-Configuration and the ID-Codes.
 4. Now you can determine the profile either by Table 47 in case of a standard slave (ID Code ≠ 0xA) or by Table 48 in case of a slave supporting extended addressing (ID Code = 0xA). The table delivers the denomination of the profile under which you can obtain more information in reference 4. Combinations for which no profile is available should not occur practically.
- ⇒ You are now aware of the profile of the slave and you are able to obtain a lot of profile dependent information about the AS-Interface Slave. A description of all defined profiles is given in Annex A of reference 4, see there for detailed profile-specific information.

Lists of possible profiles follow

- For standard slaves

Profile name according to specification	IO	ID
Remote I/Os	0 ... 8,A,C,E	0
Free profiles for Standard Slaves	0 ... E	F
Remote I/Os with dual signals	0, 3, 8	1
Safety Sensors	0	B
Single Sensor with extended control	1	1
High speed Slave Profile for Combined Transaction type 5	6	0
Slave Profile for Combined Transaction type 1 (Analog profile)	7	1
Extended Slave Profile for Combined Transaction type 1 (Extended analog profile)	7	2
Slave Profile for Combined Transaction type 1 (Integrated Analog profile)	7	3
Extended Slave Profile for Combined Transaction type 1 (Extended Integrated Analog profile)	7	4
Combi Slave with support of combined transaction type 2	7	5
Safety Sensors with non-safe outputs	7	B
Motor Control Devices (electromechanical)	7	D
Motor Control Devices (semiconductor)	7	E
Dual Actuator with feedback	B	1
Single Actuator with monitoring	D	1

Table 50: Lists of possible Profiles for Standard Slaves

- For slaves supporting extended addressing

Profile name according to specification	IO	ID	ID2
Remote I/Os	0,1,3-9,B-E	A	0
Free profiles for Slaves in extended address mode	0 ... E	A	E
Remote I/Os with dual signals	0, 3, 7, 8, B	A	2
Single Sensor with extended control	3	A	1
Combi Slave with support of combined transaction type 2	7	A	5
4I/4O in extended addressing mode	7	A	7
Slave profile for combined transaction type 4 (single channel)	7	A	8
Slave profile for combined transaction type 4 (dual channel)	7	A	9
8I/8O in extended addressing mode	7	A	A
Slave with support of combined transaction type 2	B	A	5

Table 51: Lists of possible Profiles for Slaves supporting Extended Addressing

5.3.4 AS-Interface Slave States and State Machine

The AS-Interface Slave may have one of the following states:

- *Initializing*

This state is reached after power-up or if a reset occurs. It performs the following functions:

- Resetting the outputs.
- Resetting internal registers and flags
- Load data from non-volatile RAM (such as Slave Address, IO Configuration and ID)

If initializing succeeds, the AS-Interface Slave switches over to the *Asynchronous* state.

- *Asynchronous*

In this state, the AS-Interface Slave reads the data stream and waits for a pause. If it detects this pause, it switches over to the *Receive* state. This is done in order to synchronize the slave with the cyclic data exchange of the master.

- *Receive*

The AS-Interface Slave waits for reception of a valid start bit and reads in and checks the subsequent data telegram. In case of success, the AS-Interface Slave switches over to the *Decode* state, otherwise it switches back to the *Asynchronous* state.

- *Decode*

During this state, the AS-Interface Slave compares the slave address of the received call with its own address in order to check whether it has been called or not. If this is true, the command will be analyzed and an according response will be generated. Then the AS-Interface Slave will switch over to the *Transmit* state. Otherwise, the AS-Interface Slave will switch over to the *Wait* state.

- *Wait*

If another slave was meant in the call of the master or if the command was unknown at the AS-Interface Slave, this state is reached. Similarly to the *Asynchronous* state, the AS-Interface Slave waits for synchronization purposes. Then the AS-Interface Slave will switch over to the *Synchronous* state.

- *Transmit*

During this state, the AS-Interface Slave sends the response to the decoded call that has been generated by the slave back to the master. Then the AS-Interface Slave will switch over to the *Synchronous* state.

- *Synchronous*

In *Synchronous* state, the following is performed:

- The SYNC flag is set.
- The data stream is analyzed waiting for detection of a pause.

If it detects this pause, it switches over to the *Receive* state. Then the cycle begins again.

5.3.5 Tasks of the AS-Interface Slave

The following tasks can be decoded (within the *Decode* state) and performed by an AS-Interface Slave:

- *Data Exchange*
This task is performed in cyclic operation.
- *Write Parameter*
- *Address Assignment*
- *Write Extended ID1 Code*
- *Delete Address*
- *Reset Slave*
- *Read IO Configuration*
- *Read ID*
- *Read Extended ID1 Code*
- *Read Extended ID2 Code*
- *Read Status*
- *R1*
- *Broadcast*

For more information see the specification (Reference #3).

5.3.6 Error Handling at the Slave

The receivers in the master checks the slave response for errors and vice versa the receivers in the slaves check the master request telegram for correctness. In both cases the following kinds of transmission errors will be detected by the master or the slave, respectively.

- **Start Bit Error**
The first detected bit needs to have the value 0. This bit is required for internal reference purposes. If this rule is violated, a Start Bit Error is indicated.
- **Alternating Error**
Consecutive electrical pulse should be alternating between negative and positive polarity. If two pulses of equal polarity follow directly, an Alternating Error is indicated.
- **No_Information Error**
The receiver needs to detect pulses within a given time frame in order to be able to obtain the transmitted information. If this rule is violated, a No Information Error is indicated.
- **Parity Error**
The sum of transmitted data including the parity bit but excluding the start and stop bits must be even sum. If this rule is violated, a Parity Error is indicated.

- **End Bit Error**
If no valid end bit could be detected within the appropriate time frame according to the AS-Interface specification, an End Bit Error is indicated.
- **Length Error**
If length supervision failed, a Length Error is indicated.

For more information about these errors refer to the Complete Specification, reference 3.

If a single AS-Interface Slave has an internal error, communication of the master and all other slaves this may not have any influence on the communication of the master and all other slaves.

Network errors may be detected by a watchdog timer if present.

5.4 Data Representation

This section describes the data representation of data transferred by AS-Interface for various slave profiles.

5.4.1 Standard Slaves

A standard slave device always consumes 1 byte within the dual-port memory, regardless its real number of input/output bits.

According to Complete Specification (V2.11 and higher), analog slaves with profile S-7.1 and S-7.2 are handled as standard slaves. The data transfer of the analog values has to be done in the host system.

5.4.2 Data Representation of Slave with up to 16 Bit Signals

5.4.2.1 Slave Profile S-7.3

Every measuring value of this slave module consumes 2 byte (= 16 bit). Because a module can have up to 4 channels they can consume 8 byte within the dual-port memory. The AS-Interface Master will put these values one after the other into the dual-port memory at the defined offset address for this device.

- **Slave with Digital Values (Transparent Mode)**

These slaves transfer two byte values from / to the AS-Interface Master. Values may be counter values or digital input / outputs. The AS-Interface Master will set the input data of the digital input channel to the default value 0000h in the following cases:

- after initialization of the Master if no valid data transfer has been accomplished,
- slave is not in the list of activated slaves (LAS),
- the Master detected an toggle bit error during data transfer sequence,
- the last data transfer sequence was finished with the valid bit is set to 0,
- the overflow bit was set.

- **Slave with Analog Values**

Data of this profile are fixed with a length of 16 data bits. For actuators / sensors which need less resolution the least significant bits are filled with zeros. See following table:

Data Bits	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1
16	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
12	d	d	d	d	d	d	d	d	d	d	d	d	0	0	0	0
8	d	d	d	d	d	d	d	d	0	0	0	0	0	0	0	0

d –data bit

0 – unused

Data bit D16 represents the sign bit.

The AS-Interface Master will set the input data of an analog input channel to the default value 0x7FFF, in the following cases:

- after initialization of the Master if no valid data transfer has been accomplished,
- slave is not in the list of activated slaves (LAS),
- the Master detected an toggle bit error during data transfer sequence,
- the last data transfer sequence was finished with the valid bit is set to 0,
- the overflow bit was set.

5.4.2.2 Slave Profile S-7.4

Slave profile S-7.4 has an identical mechanism for data transfer as profile S-7.3 for analog slaves. Additionally it defines a slave with four bit mode (4I/4O). This profile is designed for integrated support in AS-Interface Masters. In addition to profile S-7.3 it offers extended functions, among other things to load parameter sets into the slave for the operation of more complex slaves. The Master supports the following additional functions for S-7.4 Slaves:

- Read ID-String
- Read Parameter String
- Write Parameter String

Every measuring value of this slave module (in analog mode) consumes 2 byte (= 16 bit). Because a module can have up to 4 measuring channels it can consume 8 byte within the dual-port memory. The AS-Interface Master will put this values one after the other into the dual-port memory at the defined offset address for this device, starting with channel 0.

For analog slaves according to profile S-7.4 the transfer of analog data is handled identically to the data transfer of analog slave profile S-7.3.

Instead of transferring analog values with profile S-7.4 is optionally possible to transfer digital 16 bit input or output values. In this case the E-type in the ID-String of this Slave is set to 2.

The default value for a channel analog input or for a 16 bit digital input slave of profile S-7.4 is 7FFFh. This Value is set by the AS-Interface Master in the input process data of this channel in the following cases:

- after initialization of the Master if no valid data transfer has been accomplished,
- slave is not in the list of activated slaves (LAS),
- the Master detected an toggle bit error during data transfer sequence,
- the last data transfer sequence was finished with the valid bit is set to 0,
- the overflow bit was set

5.4.3 Slave Profile S-0.A to S-F.A

Slaves with one of these profiles can be connected in pairs of two to the AS-Interface network. Thus a maximum of 62 slaves with these profiles may operate in one network. Only slaves with ID-Code 'A' can be used as address pair. A slave pair is one slave with address 1 to 31 and one slave with first address plus 31. So for example slave address 1 and slave address 32 are a pair of slaves.

The following restrictions and features apply to these slaves:

- Each slave has only 3 parameter bits; the fourth parameter bit is unused.
- Each slave can have up to 3 bits of output data; the maximum input data of 4 bits is unchanged.
- The highest slave address is 62, the lowest address is 1.
- If there is a pair of these slaves in the AS-Interface network, the Master will exchange data with each slave every other bus-cycle. In the first cycle the Master will exchange data with the first slave, in the second cycle the Master will exchange data with second slave, in the third cycle with the first slave and so on.
- If a slave is as single slave in the AS-Interface network, the Master will exchange data with the slave in each cycle.

The highest slave address for slaves with an ID-Code unequal to 'A' remains 31.



Note: The AS-Interface specification defines the address of extended slaves with 'A' and 'B'. For compatibility, the slave addresses are numbered from 0 to 62 in the Master. So for example Slave addresses 4 and 35 mean slave address 4A and 4B in extended addressing mode.

5.5 AS-Interface and the ISO/OSI Layer Model

The AS-Interface Master protocol stack does not affect the layers 3, 4, 5 and 6 of the ISO/OSI reference model of data communication within a network.

It only specifies the following layers:

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

Above layer 7 the user area begins.

Creation of data telegram, start bit and stop bit and also protection and error processing are located at layer 2.

The `ASIMASTER` task is located on and above layer 7.

6 The Application Interface

This chapter defines the application interface of the AS-Interface 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 shall keep track of its incoming packets. It provides the communication channel for the underlying AS-Interface Master stack. Once, the AS-Interface Master stack communication is established, events received by the stack are mapped to packets that are sent to the AP-Task's process queue. Every packet has to be evaluated in the AP-Task's context and corresponding actions be executed. Additionally, Initiator-Services that are to be requested by the application are sent via predefined queue macros to the underlying AS-Interface Master stack queues via packets as well.

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

6.1 The ASIMASTER -Task

The ASIMASTER -Task coordinates, within the AS-Interface Master stack, the overlaying objects.

It is responsible for all application interactions and represents the counterpart of the AP-Task within the existing AS-Interface Master stack implementation.

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

ASCII Queue name	Description
"QUE_ASIMASTER"	Name of the ASI Master Task process queue

Table 52: ASIMASTER -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 ASIMASTER -Task. This handle is the same handle that has to be used in conjunction with the macros `TLR_QUE_SENDBUFFER_FIFO/LIFO()` for sending a packet to the ASIMASTER -Task.

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

No. of section	Packets	Page
6.1.1	ASI_MASTER_INITIALIZE_REQ/CNF – Initialize Master	98
6.1.2	ASI_MASTER_REGISTER_REQ/CNF – Register at ASi Master	100
6.1.3	ASI_MASTER_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle	103
6.1.4	ASI_MASTER_SET_SLAVE_PARAM_REQ/CNF – Set Slave Parameters	106
6.1.5	ASI_MASTER_GET_SLAVE_PARAM_REQ/CNF – Get Slave Parameter	110
6.1.6	ASI_MASTER_SET_BUS_PARAM_REQ/CNF – Set Bus Parameters	117
6.1.7	ASI_MASTER_GET_BUS_PARAM_REQ/CNF – Get Bus Parameters	121
6.1.8	ASI_MASTER_SET_OFFLINE_MODE_REQ/CNF – Set Offline Mode	123
6.1.9	ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF – Read Configuration	126
6.1.9	ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF – Read Configuration	126
6.1.11	ASI_MASTER_GET_PERMANENT_CONFIG_REQ/CNF – Get Permanent Configuration	135
6.1.12	ASI_MASTER_GET_PERMANENT_PARAMETER_REQ/CNF – Get Permanent Parameter	139
6.1.13	ASI_MASTER_GET_SLAVE_DIAG_REQ/CNF – Get Slave Diagnosis	143
6.1.14	ASI_MASTER_EXECUTE_COMMAND_REQ/CNF – Execute Command (Send a single Command to the Slave)	147
6.1.15	ASI_MASTER_WRITE_PARAMETER_REQ/CNF – Write Parameter to Slave	151
6.1.16	ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ/CNF – Change Slave Address	155
6.1.17	ASI_MASTER_WRITE_EXT_ID1_CODE_REQ/CNF – Write Extended ID 1 Code	158
6.1.18	ASI_MASTER_GET_LPS_REQ /CNF – Get List of Projected Slaves	161

6.1.19	ASI_MASTER_GET_LAS_REQ/CNF – Get List of Activated Slaves	164
6.1.20	ASI_MASTER_GET_LDS_REQ/CNF – Get List of Detected Slaves	167
6.1.21	ASI_MASTER_GET_LPF_REQ/CNF – Get List of Peripheral Faults	170
6.1.22	ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_REQ/CNF – Get Execution Control Flags	173
6.1.23	ASI_MASTER_READ_IDENTIFICATION_STRING_REQ/CNF – Read Identification String	178
6.1.24	ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ/CNF – Read Diagnostic String	185
6.1.25	ASI_MASTER_READ_PARAMETER_STRING_REQ/CNF – Read Parameter String	190
6.1.26	ASI_MASTER_WRITE_PARAMETER_STRING_REQ/CNF – Write Parameter String	195
6.1.27	ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode	198
6.1.28	ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF – Set Data Exchange Active	201
6.1.29	ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ/CNF – Set Auto Address Enable	204
6.1.30	ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_REQ/CNF – Get Auto Address Enable	207
6.1.31	ASI_MASTER_STATE_CHANGE_IND/RES – State Change Indication	210

Table 53: Topics of ASIMASTER -Task and associated packets

6.1.1 ASI_MASTER_INITIALIZE_REQ/CNF – Initialize Master

This packet is used to initialize the AS-Interface Master.



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

```

/*****
** type of <code>ASI_MASTER_PACKET_INITIALIZE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_INITIALIZE_REQ_Ttag
    ASI_MASTER_PACKET_INITIALIZE_REQ_T;

struct ASI_MASTER_PACKET_INITIALIZE_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header.        */
};
/*****

```

Packet Description

structure ASI_MASTER_PACKET_INITIALIZE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005200	ASI_MASTER_INITIALIZE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 54: ASI_MASTER_PACKET_INITIALIZE_REQ_T – Initialize Master

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_INITIALIZE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_INITIALIZE_CNF_Ttag
    ASI_MASTER_PACKET_INITIALIZE_CNF_T;

struct ASI_MASTER_PACKET_INITIALIZE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header.          */
};
/*****
    
```

Packet Description

structure ASI_MASTER_PACKET_INITIALIZE_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	Description			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005201	ASI_MASTER_INITIALIZE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change

Table 55: ASI_MASTER_PACKET_INITIALIZE_CNF_T –Confirmation for Initialize Master

6.1.2 ASI_MASTER_REGISTER_REQ/CNF – Register at ASi Master

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 such as diagnostic indications.



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

```

/*****
/** type of <code>ASI_MASTER_APP_REGISTER_REQ_DATA_Ttag</code> */
typedef struct ASI_MASTER_APP_REGISTER_REQ_DATA_Ttag
    ASI_MASTER_APP_REGISTER_REQ_DATA_T;

struct ASI_MASTER_APP_REGISTER_REQ_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

/** type of <code>ASI_MASTER_PACKET_APP_REGISTER_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_APP_REGISTER_REQ_Ttag
    ASI_MASTER_PACKET_APP_REGISTER_REQ_T;

struct ASI_MASTER_PACKET_APP_REGISTER_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;    /** packet header.      */
    ASI_MASTER_APP_REGISTER_REQ_DATA_T tData; /** packet request data. */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_APP_REGISTER_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005202	ASI_MASTER_REGISTER_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_APP_REGISTER_REQ_DATA_T			
	ulReserved	UINT32		Reserved

Table 56: ASI_MASTER_PACKET_APP_REGISTER_REQ_T – Register at ASi Master Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_APP_REGISTER_CNF_DATA_Ttag</code> */
typedef struct ASI_MASTER_APP_REGISTER_CNF_DATA_Ttag
    ASI_MASTER_APP_REGISTER_CNF_DATA_T;

struct ASI_MASTER_APP_REGISTER_CNF_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

/** type of <code>ASI_MASTER_PACKET_APP_REGISTER_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_APP_REGISTER_CNF_Ttag
    ASI_MASTER_PACKET_APP_REGISTER_CNF_T;

struct ASI_MASTER_PACKET_APP_REGISTER_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead; /** packet header.          */
    ASI_MASTER_APP_REGISTER_CNF_DATA_T tData; /** packet confirmation data. */
};
/*****
    
```

Packet Description

structure ASI_MASTER_PACKET_APP_REGISTER_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		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	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005203	ASI_MASTER_REGISTER_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	
tData	structure ASI_MASTER_APP_REGISTER_CNF_DATA_T			
	ulReserved	UINT32		

Table 57: ASI_MASTER_PACKET_APP_REGISTER_CNF_T –Confirmation for Register at ASi Master Request

6.1.3 ASI_MASTER_GET_BUFFER_HANDLE_REQ/CNF – Get Buffer Handle

This packet can be used to obtain handle to the send and receive buffer.



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

```

/*****
** type of <code>ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_Ttag
    ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_T;

struct ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header.          */
};
/*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005204	ASI_MASTER_GET_BUFFER_HANDLE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch	

Table 58: ASI_MASTER_PACKET_GET_BUFFER_HANDLE_REQ_T – Get Buffer Handle Request

Packet Structure Reference

```
/*
*****
** type of <code>ASI_MASTER_GET_BUFFER_HANDLE_CNF_DATA_Ttag</code> */
typedef struct ASI_MASTER_GET_BUFFER_HANDLE_CNF_DATA_Ttag
    ASI_MASTER_GET_BUFFER_HANDLE_CNF_DATA_T;

#define ASI_MASTER_SEND_RECV_BUFFER_SIZE 3584

struct ASI_MASTER_GET_BUFFER_HANDLE_CNF_DATA_Ttag
{
    TLR_UINT32 ulRecvBuffer;
    TLR_UINT32 ulSendBuffer;
};

** type of <code>ASI_MASTER_PACKET_GET_BUFFER_HANDLE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_BUFFER_HANDLE_CNF_Ttag
    ASI_MASTER_PACKET_GET_BUFFER_HANDLE_CNF_T;

struct ASI_MASTER_PACKET_GET_BUFFER_HANDLE_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;    /** packet header.          */
    ASI_MASTER_GET_BUFFER_HANDLE_CNF_DATA_T tData; /** packet request data.    */
};
*****
*/
```

Packet Description

structure ASI_MASTER_PACKET_GET_BUFFER_HANDLE_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		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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005205	ASI_MASTER_GET_BUFFER_HANDLE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_BUFFER_HANDLE_CNF_DATA_T			
	ulRecvBuffer	UINT32		Handle to receive buffer
	ulSendBuffer	UINT32		Handle to send buffer

Table 59: ASI_MASTER_PACKET_GET_BUFFER_HANDLE_CNF_T –Confirmation for Get Buffer Handle Request

6.1.4 ASI_MASTER_SET_SLAVE_PARAM_REQ/CNF – Set Slave Parameters

This packet is used by the AP-task to set slave parameters as a single block. The structure of the slave parameter header and data are described in section ASI_MASTER_GET_SLAVE_PARAM_REQ/CNF – Get Slave Parameter of this document.

Packet Structure Reference

```

/*****
typedef struct ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
    ASI_MASTER_SLAVE_PARAM_HEADER_T;

#define ASI_MASTER_DEVICE_FLAG_IGNORE_PP            0x01
#define ASI_MASTER_DEVICE_FLAG_IGNORE_PCD         0x02
#define ASI_MASTER_DEVICE_FLAG_IGNORE_LPS         0x04
#define ASI_MASTER_DEVICE_FLAG_RESTART            0x08
#define ASI_MASTER_DEVICE_FLAG_ACTIVE_SLAVE       0x80

#define ASI_MASTER_VESION_FLAG_CONFIG_ID1_ID2      0x04

struct ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
{
    /* general device description section */
    TLR_UINT16 usDataSetLen;
    TLR_UINT8  bDevFlag;
    TLR_UINT8  bParam;
    TLR_UINT8  bIoCode;
    TLR_UINT8  bIdCode;
    TLR_UINT8  bId1Code;
    TLR_UINT8  bId2Code;

    TLR_UINT8  bVersionFlags;

    TLR_UINT8  abOctet[7];
};

typedef struct ASI_MASTER_SLAVE_PARAM_Ttag
    ASI_MASTER_SLAVE_PARAM_T;

#define ASI_MASTER_ACTION_SET_SLAVE_PARAM          0x01
#define ASI_MASTER_ACTION_CLEAR_SLAVE_PARAM       0x02
#define ASI_MASTER_ACTION_CLEAR_ALL_SLAVE_PARAM   0x03

struct ASI_MASTER_SLAVE_PARAM_Ttag
{
    TLR_UINT8  bNotUsed;
    TLR_UINT8  bSlaveAddr;
    TLR_UINT8  bAction;
    TLR_UINT8  bReserved;

    ASI_MASTER_SLAVE_PARAM_HEADER_T tSlaveParamHeader;

    TLR_UINT8  abParamData[sizeof(ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T)+
                           sizeof(ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T)];
};

#define ASI_MASTER_SLAVE_PARAM_PRE_HEADER_SIZE    \
    (sizeof(TLR_UINT8) * 4)

#define ASI_MASTER_SLAVE_PARAM_MIN_PACKET_SIZE   \
    (ASI_MASTER_SLAVE_PARAM_PRE_HEADER_SIZE +    \
     sizeof(ASI_MASTER_SLAVE_PARAM_HEADER_T)+   \
     sizeof(TLR_UINT8)* 6)

```

```
#define ASI_MASTER_SLAVE_MAX_PARAM_SIZE \
    sizeof(ASI_MASTER_SLAVE_PARAM_T)

#define ASI_MASTER_SLAVE_MIN_PARAM_SIZE \
    (sizeof(ASI_MASTER_SLAVE_PARAM_HEADER_T)+ \
    sizeof(TLR_UINT16))

typedef struct ASI_MASTER_SET_SLAVE_PARAM_REQ_DATA_Ttag
    ASI_MASTER_SET_SLAVE_PARAM_REQ_DATA_T;

struct ASI_MASTER_SET_SLAVE_PARAM_REQ_DATA_Ttag
{
    ASI_MASTER_SLAVE_PARAM_T tSlaveParam;
};

typedef struct ASI_MASTER_PACKET_SET_SLAVE_PARAM_REQ_Ttag
    ASI_MASTER_PACKET_SET_SLAVE_PARAM_REQ_T;

struct ASI_MASTER_PACKET_SET_SLAVE_PARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead; /** packet header */
    ASI_MASTER_SET_SLAVE_PARAM_REQ_DATA_T tData; /** packet data */
};
/*****/
```

Packet Description

structure ASI_MASTER_PACKET_SET_SLAVE_PARAM_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32		Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005206	ASI_MASTER_SET_SLAVE_PARAM_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_SET_SLAVE_PARAM_REQ_DATA_T			
	tSlaveParam	ASI_MASTER_SLAVE_PARAM_T		Structure containing the slave parameters to be set.

Table 60: ASI_MASTER_PACKET_SET_SLAVE_PARAM_REQ_T – Set Slave Parameters

Packet Structure Reference

```

/*****
typedef struct ASI_MASTER_PACKET_SET_SLAVE_PARAM_CNF_Ttag
    ASI_MASTER_PACKET_SET_SLAVE_PARAM_CNF_T;

struct ASI_MASTER_PACKET_SET_SLAVE_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead; /** packet header */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_SET_SLAVE_PARAM_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	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	0 ... 2 ³² -1	Packet Identification, unchanged
		ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
		ulCmd	UINT32	0x00005207	ASI_MASTER_SET_SLAVE_PARAM_CNF - Command
		ulExt	UINT32	0	Extension, reserved
		ulRout	UINT32	x	Routing information, do not change

Table 61: ASI_MASTER_PACKET_SET_SLAVE_PARAM_CNF_T – Confirmation for Set Slave Parameters

6.1.5 ASI_MASTER_GET_SLAVE_PARAM_REQ/CNF – Get Slave Parameter

This packet is used by the AP-task to retrieve a structure containing slave parameters.

The parameter data delivered by the confirmation packet consist of the data type part and the offset descriptor part.

The slave parameter structure `ASI_MASTER_SLAVE_PARAM_T` is structured as follows:

Variable	Type	Value / Range	Description
bNotUsed	UINT8	0..255, set to 0	Not used
bSlaveAddr	UINT8	0..62	Slave address
bAction	UINT8	1..3	Action
		1	Set Slave Parameter
		2	Clear Slave Parameter
		3	Clear All Slave Parameter
bReserved	UINT8	0..255, set to 0	Reserved
tSlaveParamHeader	ASI_MASTER_SLAVE_PARAM_HEADER_T		Slave parameter header
abParamData[...]	UINT8[]		Slave parameter data

Table 62: Structure `ASI_MASTER_SLAVE_PARAM_T`

The following values are allowed for variable `bAction`:

Value of bAction	Action
1	Set Slave Parameter
2	Clear Slave Parameter
3	Clear All Slave Parameter

Table 63: Meaning of allowed Values of variable `bAction`

The slave parameter header looks as follows:

```

/*****
ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
  ASI_MASTER_SLAVE_PARAM_HEADER_T;

#define ASI_MASTER_DEVICE_FLAG_IGNORE_PP          0x01
#define ASI_MASTER_DEVICE_FLAG_IGNORE_PCD        0x02
#define ASI_MASTER_DEVICE_FLAG_IGNORE_LPS        0x04
#define ASI_MASTER_DEVICE_FLAG_RESTART           0x08
#define ASI_MASTER_DEVICE_FLAG_ACTIVE_SLAVE      0x80

#define ASI_MASTER_VESION_FLAG_CONFIG_ID1_ID2    0x04

struct ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
{
  TLR_UINT16  usDataSetLen;
  TLR_UINT8   bDevFlag;
  TLR_UINT8   bParam;
  TLR_UINT8   bIoCode;
  TLR_UINT8   bIdCode;
  TLR_UINT8   bId1Code;
  TLR_UINT8   bId2Code;

  TLR_UINT8   bVersionFlags;

  TLR_UINT8   bConfigFlags;
  TLR_UINT8   abOctet[6];
};
    
```

The slave parameter header contains the slave parameters already described in detail within section *Detailed Description of Slave Parameters* of this document.

The slave parameter data `abParamData[]` consists of a data type part and offset descriptor part.

The data type part of the slave parameter data is structured like:

Variable	Type	Value / Range	Description
<code>usDataTypeDescrLen</code>	UINT16	0..65535	The length of the array <code>ausDataType[.]</code> including the length of the parameter <code>usDataTypeDescrLen</code> itself. The length shall be specified as a length in octets.
<code>ausDataType[4]</code>	UINT16		Structure for process data related information

Table 64: Structure `ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T`

The array `ausDataType [..]` is used in order to arrange the process data modules (input/output) for the corresponding slave device. An entry in this table has to result in a corresponding entry in the `ausIoOffset[..]` array, which contains the offset address within the dual-port memory.

There are four `ausTypeData` entries. Each `ausTypeData` entry is based on the following structure:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DIR	Reserved for further use							Process data length in multiple of bits							

Table 65: Structure of `ausTypeData` Entries

Note: Every standard slave device is treated as a 4 bit module, regardless its real number of input/output bits. A 4 bit device consumes always one octet within the dual-port memory. For each analog slave (S-7.3 or S-7.4) only one module may be configured, because an analog slave is either an input or an output slave. For an analog slave, the configured process data length can be 16, 32 or 64 bit, because each channel requires 16 bit and an analog slave can have one, two or four channels.

The offset descriptor part of the parameter data is structured like:

Variable	Type	Value / Range	Description
usOffsetDescrLen	UINT16	0.. 65535	The length of the array <code>ausOffset[.]</code> including the length of the parameter <code>usOffsetDescrLen</code> itself. The length shall be specified as a length in octets.
bInputCnt	UINT8	0.. 255	The parameter fixes the number of input description entries.
bOutputCnt	UINT8	0.. 255	The parameter fixes the number of output description entries.
ausOffset[4]	UINT16[]		Structure for offset information of process data

Table 66: Structure `ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T`

The array `ausIoOffset[.]` is used in order to specify the offset address for an input or output module of a slave device. An entry in this table has to result in a corresponding entry in the `ausDataType[.]` array, which contains the length and the direction of the module.

If the AS-Interface slave device consists of input and output modules, the first entry has to be the description for the input module. All offsets has to be configured as byte offsets, because each slave device, except an analogue device, is treated as a 4 bit module, regardless of its actual number of input/output bits.

There are four `ausIoOffset` entries. Each `ausIoOffset` entry is based on the following structure:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Reserved for further use						Byte offset within the dual-port memory									

Table 67: Structure of `ausIoOffset` Entries

Packet Structure Reference

```

typedef struct ASI_MASTER_GET_SLAVE_PARAM_REQ_DATA_Ttag
  ASI_MASTER_GET_SLAVE_PARAM_REQ_DATA_T;

struct ASI_MASTER_GET_SLAVE_PARAM_REQ_DATA_Ttag
{
  TLR_UINT32 ulAddress;
};

typedef struct ASI_MASTER_PACKET_GET_SLAVE_PARAM_REQ_Ttag
  ASI_MASTER_PACKET_GET_SLAVE_PARAM_REQ_T;

struct ASI_MASTER_PACKET_GET_SLAVE_PARAM_REQ_Ttag
{
  TLR_PACKET_HEADER_T          tHead; /** packet header */
  ASI_MASTER_GET_SLAVE_PARAM_REQ_DATA_T tData; /** packet data */
};

```

Packet Description

structure ASI_MASTER_PACKET_GET_SLAVE_PARAM_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005240	ASI_MASTER_GET_SLAVE_PARAM_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_GET_SLAVE_PARAM_REQ_DATA_T			
	ulAddress	UINT32	0 ... 62	Address of slave of which parameters are retrieved

Table 68: ASI_MASTER_PACKET_GET_SLAVE_PARAM_REQ_T - Get Slave Parameter

Packet Structure Reference

```

/*****
typedef struct ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_Ttag
    ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T;

#define ASI_MASTER_MAX_DESCR_COUNT                4

struct ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_Ttag
{
    TLR_UINT16 usOffsetDescrLen;
    TLR_UINT8  bInputCnt;
    TLR_UINT8  bOutputCnt;
    TLR_UINT16 ausOffset[ASI_MASTER_MAX_DESCR_COUNT];
};
/*****/

/*****/
typedef struct ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_Ttag
    ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T;

#define ASI_MASTER_SLAVE_PARAM_BIT_DATA_LEN_MSK  0x00FF
#define ASI_MASTER_SLAVE_PARAM_DIRECTION_MSK     0x8000

struct ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_Ttag
{
    TLR_UINT16 usDataTypeDescrLen;
    TLR_UINT16 ausDataType[ASI_MASTER_MAX_DESCR_COUNT];
};
/*****/

ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
    ASI_MASTER_SLAVE_PARAM_HEADER_T;

#define ASI_MASTER_DEVICE_FLAG_IGNORE_PP        0x01
#define ASI_MASTER_DEVICE_FLAG_IGNORE_PCD      0x02
#define ASI_MASTER_DEVICE_FLAG_IGNORE_LPS      0x04
#define ASI_MASTER_DEVICE_FLAG_RESTART         0x08
#define ASI_MASTER_DEVICE_FLAG_ACTIVE_SLAVE    0x80

#define ASI_MASTER_VESION_FLAG_CONFIG_ID1_ID2  0x04

struct ASI_MASTER_SLAVE_PARAM_HEADER_Ttag
{
    /* general device description section */
    TLR_UINT16 usDataSetLen;
    TLR_UINT8  bDevFlag;
    TLR_UINT8  bParam;
    TLR_UINT8  bIoCode;
    TLR_UINT8  bIdCode;
    TLR_UINT8  bId1Code;
    TLR_UINT8  bId2Code;

    TLR_UINT8  bVersionFlags;

    TLR_UINT8  abOctet[7];
};

typedef struct ASI_MASTER_SLAVE_PARAM_Ttag
    ASI_MASTER_SLAVE_PARAM_T;

#define ASI_MASTER_ACTION_SET_SLAVE_PARAM       0x01
#define ASI_MASTER_ACTION_CLEAR_SLAVE_PARAM     0x02
#define ASI_MASTER_ACTION_CLEAR_ALL_SLAVE_PARAM 0x03

```

```
struct ASI_MASTER_SLAVE_PARAM_Ttag
{
    TLR_UINT8    bNotUsed;
    TLR_UINT8    bSlaveAddr;
    TLR_UINT8    bAction;
    TLR_UINT8    bReserved;

    ASI_MASTER_SLAVE_PARAM_HEADER_T tSlaveParamHeader;

    TLR_UINT8    abParamData[sizeof(ASI_MASTER_SLAVE_CONFIG_DATA_TYPE_DESCR_T)+
                               sizeof(ASI_MASTER_SLAVE_CONFIG_OFFSET_DESCR_T)];
};

typedef struct ASI_MASTER_GET_SLAVE_PARAM_CNF_DATA_Ttag
    ASI_MASTER_GET_SLAVE_PARAM_CNF_DATA_T;

struct ASI_MASTER_GET_SLAVE_PARAM_CNF_DATA_Ttag
{
    ASI_MASTER_SLAVE_PARAM_T tSlaveParam;
};

typedef struct ASI_MASTER_PACKET_GET_SLAVE_PARAM_CNF_Ttag
    ASI_MASTER_PACKET_GET_SLAVE_PARAM_CNF_T;

struct ASI_MASTER_PACKET_GET_SLAVE_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead; /** packet header */
    ASI_MASTER_GET_SLAVE_PARAM_CNF_DATA_T tData; /** packet data */
};
```

Packet Description

structure ASI_MASTER_PACKET_GET_SLAVE_PARAM_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		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	14	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005241	ASI_MASTER_GET_SLAVE_PARAM_CNF - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure			
	tSlaveParam	ASI_MASTER_SLAVE_PARAM_T		Slave parameter structure. For explanation see above.

Table 69: ASI_MASTER_PACKET_GET_SLAVE_PARAM_CNF_T - Confirmation for Get Slave Parameter

6.1.6 ASI_MASTER_SET_BUS_PARAM_REQ/CNF – Set Bus Parameters

This packet is used to set the bus parameters. For this purpose one single structure of the type `ASI_MASTER_BUS_PARAM_T` is used.

The bus parameters to be set are in detail:

- Operation mode

The operation mode (variable `ulOperationMode`) can be either protected or configuration. It can also be set individually by the `ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode` packet.

- Data exchange

Data exchange mode (variable `ulDataExchange`) can be inactive, active (in online mode) or active in offline mode. It can also be set individually by the `ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF – Set Data Exchange Active` packet.

- Auto addressing mode

The auto addressing mode can be enabled or disabled. It can also be set individually by the `ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ/CNF – Set Auto Address Enable` packet.

- Process data mode

The process data mode can be set to the configured offset or a fixed offset.

- Auto clear mode

The auto clear mode determines the behavior of the AS-Interface Master, if for at least one slave an error is reported. The following four options are available:

ASI_MASTER_AUTO_CLEAR_MODE_INACTIVE = 0

The Master does not take care about the status of any connected slave devices and will continue communicating with all activated slaves.

ASI_MASTER_AUTO_CLEAR_MODE_MISSING = 1

The Master will stop communication to both channels if it detects at least one device as missing by changing to offline state.

ASI_MASTER_AUTO_CLEAR_MODE_FAULT = 2

The Master will stop communication to both channels if it detects at least one device reporting a periphery fault by changing to offline state.

ASI_MASTER_AUTO_CLEAR_MODE_MISSING_FAULT = 3

The Master will stop communication to both channels if it detects at least one device as missing or reporting a periphery fault by changing to offline state.

- Data format mode

The Data format mode can be set to specify whether the data to be entered are stored in big-endian or little-endian format.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_BUS_PARAM_Ttag</code> */
typedef struct ASI_MASTER_BUS_PARAM_Ttag
    ASI_MASTER_BUS_PARAM_T;

#define ASI_MASTER_OPERATION_MODE_CONFIGURATION        0x00000000L
#define ASI_MASTER_OPERATION_MODE_PROTECTED          0x00000001L

#define ASI_MASTER_DATA_EXCHANGE_INACTIVE            0x00000000L
#define ASI_MASTER_DATA_EXCHANGE_ACTIVE              0x00000001L
#define ASI_MASTER_DATA_EXCHANGE_OFFLINE_ACTIVE      0x00000002L

#define ASI_MASTER_AUTO_ADDRESS_DISABLE              0x00000000L
#define ASI_MASTER_AUTO_ADDRESS_ENABLE               0x00000001L

#define ASI_MASTER_PROCESS_DATA_MODE_CONFIGURED_OFFSET 0x00000000L
#define ASI_MASTER_PROCESS_DATA_MODE_FIX_OFFSET      0x00000001L

#define ASI_MASTER_AUTO_CLEAR_MODE_INACTIVE           0x00000000L
#define ASI_MASTER_AUTO_CLEAR_MODE_MISSING           0x00000001L
#define ASI_MASTER_AUTO_CLEAR_MODE_FAULT             0x00000002L
#define ASI_MASTER_AUTO_CLEAR_MODE_MISSING_FAULT     0x00000003L

#define ASI_MASTER_DATA_FORMAT_MODE_BIG_ENDIAN       0x00000000L
#define ASI_MASTER_DATA_FORMAT_MODE_LITTLE_ENDIAN   0x00000001L

struct ASI_MASTER_BUS_PARAM_Ttag
{
    TLR_UINT32 ulOperationMode;
    TLR_UINT32 ulDataExchange;
    TLR_UINT32 ulAutoAddress;
    TLR_UINT32 ulProcessDataMode;
    TLR_UINT32 ulAutoClearMode;
    TLR_UINT32 ulDataFormatMode;
};

/** type of <code>ASI_MASTER_SET_BUS_PARAM_REQ_DATA_T</code> */
typedef struct ASI_MASTER_SET_BUS_PARAM_REQ_DATA_Ttag
    ASI_MASTER_SET_BUS_PARAM_REQ_DATA_T;

struct ASI_MASTER_SET_BUS_PARAM_REQ_DATA_Ttag
{
    ASI_MASTER_BUS_PARAM_T tBusParam;
};

/** type of <code>ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_T</code> */
typedef struct ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_Ttag
    ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_T;

struct ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead; /** packet header */
    ASI_MASTER_SET_BUS_PARAM_REQ_DATA_T tData; /** packet data */
};
/*****

```

Packet Description

structure ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	24	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005208	ASI_MASTER_SET_BUS_PARAM_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_SET_BUS_PARAM_REQ_DATA_T			
	tBusParam	ASI_MASTER_BUS_PARAM_T		Structure containing the bus parameters to be set.

Table 70: ASI_MASTER_PACKET_SET_BUS_PARAM_REQ_T – Set Bus Parameters

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_T</code> */
typedef struct ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_Ttag
    ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_T;

struct ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;          /** packet header */
};
/*****

```

Packet Description

structure ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	Description			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005209	ASI_MASTER_SET_BUS_PARAM_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change

Table 71: ASI_MASTER_PACKET_SET_BUS_PARAM_CNF_T –Confirmation for Set Bus Parameters

6.1.7 ASI_MASTER_GET_BUS_PARAM_REQ/CNF – Get Bus Parameters

This packet is used to get the currently valid values of the bus parameters. For this purpose one single structure of the type ASI_MASTER_PACKET_GET_BUS_PARAM_T is used.

For more details concerning the single bus parameters in this structure, see the preceding section (ASI_MASTER_SET_BUS_PARAM_REQ/CNF – Set Bus Parameters).

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_T</code> */
typedef struct ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_Ttag
    ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_T;
/** type of <code>ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_T</code> */

struct ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header */
};
/*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000520A	ASI_MASTER_GET_BUS_PARAM_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 72: ASI_MASTER_PACKET_GET_BUS_PARAM_REQ_T – Get Bus Parameters

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_BUS_PARAM_CNF_DATA_T</code> */
typedef struct ASI_MASTER_GET_BUS_PARAM_CNF_DATA_Ttag
    ASI_MASTER_GET_BUS_PARAM_CNF_DATA_T;

struct ASI_MASTER_GET_BUS_PARAM_CNF_DATA_Ttag
{
    ASI_MASTER_BUS_PARAM_T tBusParam;
};

/** type of <code>ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_T</code> */
typedef struct ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_Ttag
    ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_T;

struct ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead; /** packet header */
    ASI_MASTER_GET_BUS_PARAM_CNF_DATA_T tData; /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_BUS_PARAM_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		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	24	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000520B	ASI_MASTER_GET_BUS_PARAM_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	
tData	structure ASI_MASTER_GET_BUS_PARAM_CNF_DATA_T			
	tBusParam	ASI_MASTER_BUS_PARAM_T		Structure containing the retrieved bus parameters.

Table 73: ASI_MASTER_PACKET_GET_BUS_PARAM_CNF_T – Confirmation for Get Bus Parameters

6.1.8 ASI_MASTER_SET_OFFLINE_MODE_REQ/CNF – Set Offline Mode

This service has to be used by the AP-Task in order to request the ASIMASTER-Task to leave or change to the offline phase.

After entering the offline phase, the AS-Interface Master resets all connected slaves and no further communication is possible on the AS-Interface network until the offline phase is left.

The `ulOfflineMode` parameter defines whether the ASIMASTER -Task should leave or change to the offline phase.

- For changing to the offline mode set the `ulOfflineMode` parameter to the value `ASI_MASTER_SET_OFFLINE_MODE = 1`.
- For leaving the offline mode set the `ulOfflineMode` parameter to the value `ASI_MASTER_RESET_OFFLINE_MODE = 2`.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER -Task process queue.



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

```

/*****
/** type of <code>ASI_MASTER_SET_OFFLINE_MODE_REQ_DATA_T</code> */
typedef struct ASI_MASTER_SET_OFFLINE_MODE_REQ_DATA_Ttag
    ASI_MASTER_SET_OFFLINE_MODE_REQ_DATA_T;

#define ASI_MASTER_SET_OFFLINE_MODE        0x00000001L
#define ASI_MASTER_RESET_OFFLINE_MODE     0x00000002L

struct ASI_MASTER_SET_OFFLINE_MODE_REQ_DATA_Ttag
{
    TLR_UINT32 ulOfflineMode;
};

/** type of <code>ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_Ttag
    ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_T;

struct ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_SET_OFFLINE_MODE_REQ_DATA_T tData;  /** packet data */
};

*****/

```

Packet Description

structure ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	QUE_ASIMASTER	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	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32		Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000520C	ASI_MASTER_SET_OFFLINE_MODE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_SET_OFFLINE_MODE_REQ_DATA_T			
	ulOfflineMode	UINT32	1...2	Offline mode flag indicating if set to ASI_MASTER_SET_OFFLINE_MODE (1) that the AS-Interface Master should change to the offline phase. If set to ASI_MASTER_RESET_OFFLINE_MODE (0), the AS-Interface Master should leave the offline phase.

Table 74: ASI_MASTER_PACKET_SET_OFFLINE_MODE_REQ_T – Set Offline Mode

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_SET_OFFLINE_MODE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_OFFLINE_MODE_CNF_Ttag
    ASI_MASTER_PACKET_SET_OFFLINE_MODE_CNF_T;

struct ASI_MASTER_PACKET_SET_OFFLINE_MODE_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead;  /** packet header. */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_SET_OFFLINE_MODE_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		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	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000520D	ASI_MASTER_SET_OFFLINE_MODE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change

Table 75: ASI_MASTER_PACKET_SET_OFFLINE_MODE_CNF_T – Confirmation for Set Offline Mode

6.1.9 ASI_MASTER_READ_ACTUAL_CONFIG_REQ/CNF – Read Configuration

This service has to be used by the AP-Task in order to get the configuration data image (CDI) of the AS-Interface network from the ASIMASTER-Task.

The configuration data image contains

- the current IO-Configuration,
- the ID-Code,
- the Extended ID1-Code
- and the Extended ID2-Code,

determined by reading data from the corresponding slave. These are stored in the structure `ASI_MASTER_SLAVE_CONFIG_DATA_T` within the confirmation packet:

Variable	Type	Value / Range	Description
<code>bloConfig</code>	UINT8	0..15	The current IO-Configuration
<code>bldCode</code>	UINT8	0..15	ID-Code
<code>bld1Code</code>	UINT8	0..15	Extended ID1-Code
<code>bld2Code</code>	UINT8	0..15	Extended ID2-Code

Table 76: Configuration Data Structure `ASI_MASTER_SLAVE_CONFIG_DATA_T`

If no slave is detected at specific address, all configuration data value for this slave is set to the default value (0x0F).

In section *Relation between IO Config, ID Codes and Slave Profiles* on page 85 of this document you can read about the relation of IO Configuration and the ID Codes to the slave profile and how to derive the appropriate slave profile associated to a certain slave containing a lot of information about this slave from the values delivered by the confirmation packet.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER-Task process queue.

Packet Structure Reference

```
/******  
typedef struct ASI_MASTER_READ_ACTUAL_CONFIG_REQ_DATA_Ttag  
    ASI_MASTER_READ_ACTUAL_CONFIG_REQ_DATA_T;  
  
struct ASI_MASTER_READ_ACTUAL_CONFIG_REQ_DATA_Ttag  
{  
    TLR_UINT32 ulAddress;  
};  
  
/** type of <code>ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_Ttag</code> */  
typedef struct ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_Ttag  
    ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_T;  
  
struct ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_Ttag  
{  
    TLR_PACKET_HEADER_T          tHead; /** packet header. */  
    ASI_MASTER_READ_ACTUAL_CONFIG_REQ_DATA_T tData; /** packet data */  
};  
/******
```

Packet Description

structure ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005210	ASI_MASTER_READ_ACTUAL_CONFIG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_READ_ACTUAL_CONFIG_REQ_DATA_T			
	ulAddress	UINT32	0 ... 62, 255	Address of slave (0..62). The value 255 means <i>Read configuration of all slaves.</i>

Table 77: ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_REQ_T – Read Configuration

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_SLAVE_CONFIG_DATA_Ttag</code> */
typedef struct ASI_MASTER_SLAVE_CONFIG_DATA_Ttag
    ASI_MASTER_SLAVE_CONFIG_DATA_T;

struct ASI_MASTER_SLAVE_CONFIG_DATA_Ttag
{
    TLR_UINT8 bIoConfig; /* IO-Config */
    TLR_UINT8 bIdCode; /* ID-Code */
    TLR_UINT8 bId1Code; /* ID1-Code */
    TLR_UINT8 bId2Code; /* ID2-Code */
};

/** type of <code>ASI_MASTER_READ_ACTUAL_CONFIG_CNF_DATA_T</code> */
typedef struct ASI_MASTER_READ_ACTUAL_CONFIG_CNF_DATA_Ttag
    ASI_MASTER_READ_ACTUAL_CONFIG_CNF_DATA_T;

struct ASI_MASTER_READ_ACTUAL_CONFIG_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    ASI_MASTER_SLAVE_CONFIG_DATA_T atCdi[ASI_MASTER_MAX_SLAVES];
};

/** type of <code>ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_CNF_Ttag
    ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_CNF_T;

struct ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header. */
    ASI_MASTER_READ_ACTUAL_CONFIG_CNF_DATA_T tData; /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_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		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	8 256 if ulAddress = 255	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005211	ASI_MASTER_READ_ACTUAL_CONFIG_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_READ_ACTUAL_CONFIG_CNF_DATA_T			
	ulAddress	UINT32	0 ... 62, 255	Address of slave
	atCdi[ASI_MASTER_MAX_SLAVE_S]	ASI_MASTER_SLAVE_CONFIG_DATA_T []		Array of structure for each slave

Table 78: ASI_MASTER_PACKET_READ_ACTUAL_CONFIG_CNF_T –Confirmation for Read Configuration

6.1.10 ASI_MASTER_READ_PARAMETER_IMAGE_REQ/CNF – Read Parameter Image

This packet can be used to read out the parameter image of a slave of your choice. The address of the chosen slave must be specified in variable `ulAddress` of the request packet.

The parameter image contains the parameter output of all active Slaves, determined by the last `Write_Parameter` request (see section `ASI_MASTER_WRITE_PARAMETER_REQ/CNF – Write Parameter to Slave`).

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_READ_PARAMETER_IMAGE_REQ_DATA_T</code> */
typedef struct ASI_MASTER_READ_PARAMETER_IMAGE_REQ_DATA_Ttag
    ASI_MASTER_READ_PARAMETER_IMAGE_REQ_DATA_T;

struct ASI_MASTER_READ_PARAMETER_IMAGE_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};

/** type of <code>ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_Ttag
    ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_T;

struct ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead; /** packet header. */
    ASI_MASTER_READ_PARAMETER_IMAGE_REQ_DATA_T tData; /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005212	ASI_MASTER_READ_PARAMETER_IMAGE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_READ_PARAMETER_IMAGE_REQ_DATA_T			
	ulAddress	UINT32	0 ... 62, 255	Address of the slave whose parameter image is to be read out.

Table 79: ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_REQ_T –Read Parameter Image

Packet Structure Reference

```
/*
** type of <code>ASI_MASTER_READ_PARAMETER_IMAGE_CNF_DATA_T</code> */
typedef struct ASI_MASTER_READ_PARAMETER_IMAGE_CNF_DATA_Ttag
    ASI_MASTER_READ_PARAMETER_IMAGE_CNF_DATA_T;

struct ASI_MASTER_READ_PARAMETER_IMAGE_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 aulPi[ASI_MASTER_MAX_SLAVES];
};

/*
** type of <code>ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_CNF_Ttag
    ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_CNF_T;

struct ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header. */
    ASI_MASTER_READ_PARAMETER_IMAGE_CNF_DATA_T tData; /** packet data */
};
*/
```

Packet Description

structure ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_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		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	8 256 if ulAddress = 255	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005213	ASI_MASTER_READ_PARAMETER_IMAGE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_READ_PARAMETER_IMAGE_CNF_DATA_T			
	ulAddress	UINT32	0 ... 62, 255	Address of the slave whose parameter image is to be read out.
	aulPi[ASI_MASTER_MAX_SLAVES]	UINT32[]		Array containing the requested parameter image.

Table 80: ASI_MASTER_PACKET_READ_PARAMETER_IMAGE_CNF_T –Confirmation for Read Parameter Image

6.1.11 ASI_MASTER_GET_PERMANENT_CONFIG_REQ/CNF – Get Permanent Configuration

This service has to be used by the AP-Task in order to get the permanent configuration data (PCD) and the permanent parameter (PP) of the AS-Interface network from the ASIMASTER -Task.

The permanent configuration data contains the configured IO-Configuration, ID-Code, Extended ID1-Code and Extended ID2-Code, determined by the local configuration of the ASIMASTER Task. If no Slave is configured at specific address, all configuration data value for this Slave is set to default value (0x0F).

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER -Task process queue.

The returned structure `atPcd[]` within the confirmation packet looks like:

Variable	Type	Value / Range	Description
<code>bIoConfig</code>	UINT8	0..15	The current IO-Configuration
<code>bIdCode</code>	UINT8	0..15	ID-Code
<code>bId1Code</code>	UINT8	0..15	Extended ID1-Code
<code>bId2Code</code>	UINT8	0..15	Extended ID2-Code

Table 81: Configuration Data Structure `ASI_MASTER_SLAVE_CONFIG_DATA_T`

Packet Structure Reference

```

/*****
/*****
/** type of <code>ASI_MASTER_GET_PERMANENT_CONFIG_REQ_DATA_T</code> */
typedef struct ASI_MASTER_GET_PERMANENT_CONFIG_REQ_DATA_Ttag
    ASI_MASTER_GET_PERMANENT_CONFIG_REQ_DATA_T;

struct ASI_MASTER_GET_PERMANENT_CONFIG_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};

/** type of <code>ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_Ttag
    ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_T;

struct ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;    /** packet header. */
    ASI_MASTER_GET_PERMANENT_CONFIG_REQ_DATA_T tData; /** packet data */
};
/*****

```

Packet Description

structure ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005214	ASI_MASTER_GET_PERMANENT_CONFIG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_GET_PERMANENT_CONFIG_REQ_DATA_T			
	ulAddress	UINT32	0 ... 62, 255	Address of slave

Table 82: ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_REQ_T – Get Permanent Configuration Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_SLAVE_CONFIG_DATA_Ttag</code> */
typedef struct ASI_MASTER_SLAVE_CONFIG_DATA_Ttag
    ASI_MASTER_SLAVE_CONFIG_DATA_T;

struct ASI_MASTER_SLAVE_CONFIG_DATA_Ttag
{
    TLR_UINT8 bIoConfig; /* IO-Config */
    TLR_UINT8 bIdCode; /* ID-Code */
    TLR_UINT8 bId1Code; /* ID1-Code */
    TLR_UINT8 bId2Code; /* ID2-Code */
};

/** type of <code>ASI_MASTER_GET_PERMANENT_CONFIG_CNF_DATA_T</code> */
typedef struct ASI_MASTER_GET_PERMANENT_CONFIG_CNF_DATA_Ttag
    ASI_MASTER_GET_PERMANENT_CONFIG_CNF_DATA_T;

struct ASI_MASTER_GET_PERMANENT_CONFIG_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    ASI_MASTER_SLAVE_CONFIG_DATA_T atPcd[ASI_MASTER_MAX_SLAVES];
};

/** type of <code>ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_Ttag
    ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_T;

struct ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header. */
    ASI_MASTER_GET_PERMANENT_CONFIG_CNF_DATA_T tData; /** packet data */
};
/*****

```

Packet Description

structure ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_T				
Type: Confirmation				
Area	Area	Area	Area	Area
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	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	8 256 if ulAddress = 255	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005215	ASI_MASTER_GET_PERMANENT_CONFIG_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_PERMANENT_CONFIG_CNF_DATA_T			
	ulAddress	UINT32	0 ... 62, 256	Address of slave
	atPcd[ASI_MASTER_MAX_SLAVE_S]	ASI_MASTER_SLAVE_CONFIG_DATA_T []		Array of structures containing configuration data for each slave

Table 83: ASI_MASTER_PACKET_GET_PERMANENT_CONFIG_CNF_T –Confirmation for Get Permanent Configuration Request

6.1.12 ASI_MASTER_GET_PERMANENT_PARAMETER_REQ/CNF – Get Permanent Parameter

This packet can be used to retrieve the permanent parameters for a slave of your choice.

The permanent parameter contains the configured parameter output of all AS-Interface Slaves determined by the local configuration of the ASIMASTER -Task.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_PERMANENT_PARAMETER_REQ_DATA_T</code> */
typedef struct ASI_MASTER_GET_PERMANENT_PARAMETER_REQ_DATA_Ttag
    ASI_MASTER_GET_PERMANENT_PARAMETER_REQ_DATA_T;

struct ASI_MASTER_GET_PERMANENT_PARAMETER_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};

/** type of <code>ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_Ttag
    ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_T;

struct ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header.    */
    ASI_MASTER_GET_PERMANENT_PARAMETER_REQ_DATA_T tData; /** packet data    */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005216	ASI_MASTER_GET_PERMANENT_PARAMETER_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_GET_PERMANENT_PARAMETER_REQ_DATA_T			
	ulAddress	UINT32	0 ... 62, 255	Address of slave

Table 84: ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_REQ_T – Get Permanent Parameter

Packet Structure Reference

```
/*
** type of <code>ASI_MASTER_GET_PERMANENT_PARAMETER_CNF_DATA_T</code> */
typedef struct ASI_MASTER_GET_PERMANENT_PARAMETER_CNF_DATA_Ttag
    ASI_MASTER_GET_PERMANENT_PARAMETER_CNF_DATA_T;

struct ASI_MASTER_GET_PERMANENT_PARAMETER_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 aulPp[ASI_MASTER_MAX_SLAVES];
};

/*
** type of <code>ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_CNF_Ttag
    ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_CNF_T;

struct ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header. */
    ASI_MASTER_GET_PERMANENT_PARAMETER_CNF_DATA_T tData; /** packet data */
};
*/
```

Packet Description

structure ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_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		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	8 256 if ulAddress = 255	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005217	ASI_MASTER_GET_PERMANENT_PARAMETER_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_PERMANENT_PARAMETER_CNF_DATA_T			
	ulAddress	UINT32	0 ... 62, 256	Address of slave
	aulPp[ASI_MASTER_MAX_SLAVES]	UINT32[]		Array of structures containing permanent data for each slave

Table 85: ASI_MASTER_PACKET_GET_PERMANENT_PARAMETER_CNF_T – Confirmation for Get Permanent Parameter

6.1.13 ASI_MASTER_GET_SLAVE_DIAG_REQ/CNF – Get Slave Diagnosis

This packet can be used to retrieve diagnostic information from the slaves. Each slave delivers three bytes of diagnostic information. For information about the contents and structure of these three bytes refer to the slaves documentation.

The ASI_MASTER_DIAG_REQ_FLAG_PEEK flag has the following meaning:

- If set, the data are only fetched. the diagnostic information remains.
- If not set, the diagnostic information is erased (if no current diagnostic informations are present)

The variable `ulAddress` contains the address of the slave whose diagnostic informations are to be read out.

The data structure `ASI_MASTER_SLAVE_DIAG_T` which is returned within the confirmation packet looks like:

Variable	Type	Value / Range	Description
<code>ulDiagFlags</code>	UINT32	0.. 1	Diagnostic flags
<code>ulNumOfDiagInfos</code>	UINT32	0..8	Number of currently present diagnostic informations
<code>aeSlaveError[ASI_MASTER_SLAVE_DIAG_BUFFER_SIZE]</code>	TLR_RES ULT[8]		Field with 8 entries containing the error codes of the last recently occurred errors

Table 86: Data structure `ASI_MASTER_SLAVE_DIAG_T`

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_SLAVE_DIAG_REQ_DATA_Ttag</code> */
typedef struct ASI_MASTER_GET_SLAVE_DIAG_REQ_DATA_Ttag
    ASI_MASTER_GET_SLAVE_DIAG_REQ_DATA_T;

#define ASI_MASTER_DIAG_REQ_FLAG_PEEK                0x00000001L

struct ASI_MASTER_GET_SLAVE_DIAG_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 ulReqFlags;
};

typedef struct ASI_MASTER_PACKET_GET_SLAVE_DIAG_REQ_Ttag
    ASI_MASTER_PACKET_GET_SLAVE_DIAG_REQ_T;

struct ASI_MASTER_PACKET_GET_SLAVE_DIAG_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_GET_SLAVE_DIAG_REQ_DATA_T  tData;  /** packet data */
};
/*****/

```

Packet Description

structure ASI_MASTER_GET_SLAVE_DIAG_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000521A	ASI_MASTER_GET_SLAVE_DIAG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_GET_SLAVE_DIAG_REQ_DATA_T			
	ulAddress	UINT32	1 ... 62	Address of slave
	ulReqFlags	UINT32		Request flag

Table 87: ASI_MASTER_GET_SLAVE_DIAG_REQ_T – Get Slave Diagnosis

Packet Structure Reference

```

/*****
typedef struct ASI_MASTER_SLAVE_DIAG_Ttag
    ASI_MASTER_SLAVE_DIAG_T;

#define ASI_MASTER_SLAVE_DIAG_BUFFER_SIZE                8
#define ASI_MASTER_DIAG_FLAG_DIAG_INFORMATION_LOST      0x00000001L

struct ASI_MASTER_SLAVE_DIAG_Ttag
{
    TLR_UINT32 ulDiagFlags;
    TLR_UINT32 ulNumOfDiagInfos;
    TLR_RESULT aeSlaveError[ASI_MASTER_SLAVE_DIAG_BUFFER_SIZE];
};

typedef struct ASI_MASTER_GET_SLAVE_DIAG_CNF_DATA_Ttag
    ASI_MASTER_GET_SLAVE_DIAG_CNF_DATA_T;

struct ASI_MASTER_GET_SLAVE_DIAG_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 ulReqFlags;
    ASI_MASTER_SLAVE_DIAG_T    tSlaveDiag;
};

/** type of <code>ASI_MASTER_PACKET_GET_SLAVE_DIAG_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_SLAVE_DIAG_CNF_Ttag
    ASI_MASTER_PACKET_GET_SLAVE_DIAG_CNF_T;

struct ASI_MASTER_PACKET_GET_SLAVE_DIAG_CNF_Ttag
{
    TLR_PACKET_HEADER_T        tHead;  /** packet header. */
    ASI_MASTER_GET_SLAVE_DIAG_CNF_DATA_T    tData;  /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_GET_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		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	depends on number of returned diagnostic data	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000521B	ASI_MASTER_GET_SLAVE_DIAG_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_SLAVE_DIAG_CNF_DATA_T			
	ulAddress	UINT32	1 ... 63	Address of slave
	ulReqFlags	UINT32		Request flag
	tSlaveDiag	ASI_MASTER_SLAVE_DIAG_T		Structure for slave diagnosis

Table 88: ASI_MASTER_PACKET_GET_SLAVE_DIAG_CNF_T –Confirmation for Get Slave Diagnosis

6.1.14 ASI_MASTER_EXECUTE_COMMAND_REQ/CNF – Execute Command (Send a single Command to the Slave)

This service has to be used by the AP-Task in order to request the ASIMASTER -Task to send a single AS-Interface command to a specific address for being processed.

The `ulAddr` parameter defines the address of the requested AS-Interface Slave. This address is a value in the range between 0 and 31, independently whether it is the address of a standard slave or an extended slave A or B.

The `ulInfo` parameter depends on the AS-Interface command to be sent. This parameter also has a value in the range between 0 and 31. It may, however, also contain data or parameter information.

The possible values for the `ulInfo` parameter depending on the desired AS-Interface command are given in the table below:

AS-Interface command	ulInfo parameter Standard addressing	ulInfo parameter Extended addressing A-Slave	ulInfo parameter Extended addressing B-Slave
Data Exchange (currently not supported)	0..15	8..15	0..7
Write Parameter (currently not supported)	16..31	24..31	16..23
Address Assignment (currently not supported)	0..31		
Write Extended ID1 Code	0..15		
Delete Address (currently not supported)	0	8	0
Reset Slave	28	28	20
Read IO-Configuration of Slave	16	16	24
Read ID-Code of Slave	17	17	25
Read Extended ID1- Code of Slave	18	18	26
Read Extended ID2- Code of Slave	19	19	27
Read status of Slave	30	30	22
R1	31	31	23
Broadcast reset	21		

Table 89: Possible Command Values for Parameter `ulInfo`

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```
/*
** type of <code>ASI_MASTER_EXECUTE_COMMAND_REQ_DATA_T</code> */
typedef struct ASI_MASTER_EXECUTE_COMMAND_REQ_DATA_Ttag
    ASI_MASTER_EXECUTE_COMMAND_REQ_DATA_T;

#define ASI_MASTER_EXECUTE_COMMAND_MIN_INFO 0x00000001L
#define ASI_MASTER_EXECUTE_COMMAND_MAX_INFO 0x0000001FL

struct ASI_MASTER_EXECUTE_COMMAND_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 ulInfo;
};

/*
** type of <code>ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_Ttag
    ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_T;

struct ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_EXECUTE_COMMAND_REQ_DATA_T tData; /** packet data */
};
*/
```

Packet Description

structure ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000521C	ASI_MASTER_EXECUTE_COMMAND_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_EXECUTE_COMMAND_REQ_DATA_T			
	ulAddress	UINT32	0 ... 31	Address of the requested slave to execute the command
	ulInfo	UINT32	0 ... 31	Command code, see <i>Table 89: Possible Command Values for Parameter ulInfo</i> . The command code may also contain data and decide between A and B slaves at extended addressing.

Table 90: ASI_MASTER_PACKET_EXECUTE_COMMAND_REQ_T – Execute Command Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_EXECUTE_COMMAND_CNF_DATA_T</code> */
typedef struct ASI_MASTER_EXECUTE_COMMAND_CNF_DATA_Ttag
    ASI_MASTER_EXECUTE_COMMAND_CNF_DATA_T;

#define ASI_MASTER_EXECUTE_COMMAND_MIN_INFO 0x00000001L
#define ASI_MASTER_EXECUTE_COMMAND_MAX_INFO 0x0000001FL

struct ASI_MASTER_EXECUTE_COMMAND_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 ulData;
};

/** type of <code>ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_Ttag
    ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_T;

struct ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_EXECUTE_COMMAND_CNF_DATA_T tData;  /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_T				
Type: Confirmation				
Area	Area	Area	Area	Area
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000521D	ASI_MASTER_EXECUTE_COMMAND_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	
tData	structure ASI_MASTER_EXECUTE_COMMAND_CNF_DATA_T			
	ulAddress	UINT32	0 ... 31	Address of the slave
	ulData	UINT32	0 ... 2 ³² -1	Response data of slave containing the result of executed command

Table 91: ASI_MASTER_PACKET_EXECUTE_COMMAND_CNF_T – Confirmation for Execute Command Request

6.1.15 ASI_MASTER_WRITE_PARAMETER_REQ/CNF – Write Parameter to Slave

This service has to be used by the AP-Task in order to request the ASIMASTER task to send an acyclic write parameter request to an activated Slave. The parameter data will be stored in the ASIMASTER tasks parameter image (PI).

The `ulAddress` parameter defines the address of the requested AS-Interface Slave.

The `ulParam` parameter defines the parameter data to be written to the slave. It is a value between 0 and 7 for extended slaves and between 0 and 15 for standard slaves.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_WRITE_PARAMETER_REQ_DATA_T</code> */
typedef struct ASI_MASTER_WRITE_PARAMETER_REQ_DATA_Ttag
    ASI_MASTER_WRITE_PARAMETER_REQ_DATA_T;

struct ASI_MASTER_WRITE_PARAMETER_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 ulParam;
};

/** type of <code>ASI_MASTER_PACKET_WRITE_PARAMETER_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_WRITE_PARAMETER_REQ_Ttag
    ASI_MASTER_PACKET_WRITE_PARAMETER_REQ_T;

struct ASI_MASTER_PACKET_WRITE_PARAMETER_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header.  */
    ASI_MASTER_WRITE_PARAMETER_REQ_DATA_T tData;  /** packet data      */
};
/*****

```

Packet Description

structure ASI_MASTER_WRITE_PARAMETER_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000521E	ASI_MASTER_WRITE_PARAMETER_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_WRITE_PARAMETER_REQ_DATA_T			
	ulAddress	UINT32	0 .. 62	Address of the requested slave
	ulParam	UINT32	0 ... 15	Parameter data to be written to slave. Values 8 to 15 can only be used for standard slaves, not for extended slaves.

Table 92: ASI_MASTER_WRITE_PARAMETER_REQ_T – Write Parameter to Slave Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_WRITE_PARAMETER_CNF_DATA_T</code> */
typedef struct ASI_MASTER_WRITE_PARAMETER_CNF_DATA_Ttag
    ASI_MASTER_WRITE_PARAMETER_CNF_DATA_T;

#define ASI_MASTER_MAX_PARAM_STD_SLAVE 0x0000000FL
#define ASI_MASTER_MAX_PARAM_EXT_SLAVE 0x00000007L

struct ASI_MASTER_WRITE_PARAMETER_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT32 ulParam;
};

/** type of <code>ASI_MASTER_PACKET_WRITE_PARAMETER_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_WRITE_PARAMETER_CNF_Ttag
    ASI_MASTER_PACKET_WRITE_PARAMETER_CNF_T;

struct ASI_MASTER_PACKET_WRITE_PARAMETER_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_WRITE_PARAMETER_CNF_DATA_T tData;  /** packet data */
};
*****/
```

Packet Description

structure ASI_MASTER_PACKET_WRITE_PARAMETER_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		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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000521F	ASI_MASTER_WRITE_PARAMETER_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_WRITE_PARAMETER_CNF_DATA_T			
	ulAddress	UINT32	0 .. 62	Address of the requested slave
	ulParam	UINT32	0 .. 15	Parameter data written to slave. Values 8 to 15 can only be used for standard slaves, not for extended slaves.

Table 93: ASI_MASTER_WRITE_PARAMETER_REQ_T – Confirmation for Write Parameter to Slave Request

6.1.16 ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ/CNF – Change Slave Address

This service has to be used by the AP-Task in order to request the ASIMASTER task to change the slave address of a specific slave address.

The `ulOldAddr` parameter defines the old i.e. the current address of the slave.

The `ulNewAddr` parameter defines the new address of the slave.

The values for parameter `ulOldAddr` and parameter `ulNewAddr` must be different.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T</code> */
typedef struct ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_Ttag
    ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T;

struct ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_Ttag
{
    TLR_UINT32 ulOldAddress;
    TLR_UINT32 ulNewAddress;
};

/** type of <code>ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_REQ_Ttag
    ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_REQ_T;

struct ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T tData; /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005220	ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T			
	ulOldAddress	UINT32	0 ... 62	Address to be changed.
	ulNewAddress	UINT32	0 ... 62	New address to be used.

Table 94: ASI_MASTER_CHANGE_SLAVE_ADDRESS_REQ_DATA_T – Change Slave Address Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_CNF_Ttag
    ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_CNF_T;

struct ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header. */
};

*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_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		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	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005221	ASI_MASTER_CHANGE_SLAVE_ADDRESS_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change

Table 95: ASI_MASTER_PACKET_CHANGE_SLAVE_ADDRESS_CNF_T –Confirmation for Change Slave Address Request

6.1.17 ASI_MASTER_WRITE_EXT_ID1_CODE_REQ/CNF – Write Extended ID 1 Code

This service has to be used by the AP-Task in order to request the ASIMASTER -Task to change the Extended ID1-Code of a slave with zero address.

The `ulExtID1Code` parameter defines the Extended ID1-Code to be written to the slave with zero address.

The upper limit of the range of allowed values for the Extended ID1-Code is:

- `ASI_MASTER_MAX_EXT_ID1_CODE_STD_SLAVE` = 15 for standard slaves
 - `ASI_MASTER_MAX_EXT_ID1_CODE_EXT_SLAVE` = 7 for extended slaves

There are two cases in which the packet cannot be successfully processed:

- There is no slave with zero address
- Write access to ID 1 Code is blocked

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER -Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_WRITE_EXT_ID1_CODE_REQ_DATA_T</code> */
typedef struct ASI_MASTER_WRITE_EXT_ID1_CODE_REQ_DATA_Ttag
  ASI_MASTER_WRITE_EXT_ID1_CODE_REQ_DATA_T;

#define ASI_MASTER_MAX_EXT_ID1_CODE_STD_SLAVE 0x0000000FL
#define ASI_MASTER_MAX_EXT_ID1_CODE_EXT_SLAVE 0x00000007L

struct ASI_MASTER_WRITE_EXT_ID1_CODE_REQ_DATA_Ttag
{
  TLR_UINT32 ulExtID1Code;
};

/** type of <code>ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_Ttag
  ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_T;

struct ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_Ttag
{
  TLR_PACKET_HEADER_T          tHead;  /** packet header. */
  ASI_MASTER_WRITE_EXT_ID1_CODE_REQ_DATA_T tData; /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005222	ASI_MASTER_WRITE_EXT_ID1_CODE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_WRITE_EXT_ID1_CODE_REQ_DATA_T			
	ulExtID1Code	UINT32	0 ... 15	Extended ID1-Code to be written to slave with zero address. Values 8 to 15 can only be used for standard slaves, not for extended slaves.

Table 96: ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_REQ_T – Write External ID 1 Code

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_CNF_Ttag
    ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_CNF_T;

struct ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header. */
};
/*****
    
```

Packet Description

structure ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_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		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005223	ASI_MASTER_WRITE_EXT_ID1_CODE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	

Table 97: ASI_MASTER_PACKET_WRITE_EXT_ID1_CODE_CNF_T – Confirmation for Write External ID 1 Code Request

6.1.18 ASI_MASTER_GET_LPS_REQ /CNF – Get List of Projected Slaves

This packet can be used to obtain a list of slaves which have been projected.

This List of Projected Slaves (LPS) is stored in variable `abProjectedList`. It consists of 8 bytes. These are represented as a bit field for the corresponding channel. The following table shows, which bit of the LPS is related to which slave.

The LPS is part of the Extended Status Block, see there.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Byte #								
1	7	6	5	4	3	2	1	0
2	15	14	13	12	11	10	9	8
3	23	22	21	20	19	18	17	16
...								
8	x	62	61	60	59	58	57	56

Table 98: Representation of List of Projected Slaves (LPS)

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means the value does not matter.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_GET_LPS_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LPS_REQ_Ttag
    ASI_MASTER_PACKET_GET_LPS_REQ_T;

struct ASI_MASTER_PACKET_GET_LPS_REQ_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LPS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005224	ASI_MASTER_GET_LPS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 99: ASI_MASTER_PACKET_GET_LPS_REQ_T – Get List of projected Slaves Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_LPS_CNF_DATA_Ttag</code> */
#define ASI_MASTER_LIST_SIZE 8

typedef struct ASI_MASTER_GET_LPS_CNF_DATA_Ttag
    ASI_MASTER_GET_LPS_CNF_DATA_T;

struct ASI_MASTER_GET_LPS_CNF_DATA_Ttag
{
    TLR_UINT8    abProjectedList[ASI_MASTER_LIST_SIZE];
};

/** type of <code>ASI_MASTER_PACKET_GET_LPS_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LPS_CNF_Ttag
    ASI_MASTER_PACKET_GET_LPS_CNF_T;

struct ASI_MASTER_PACKET_GET_LPS_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
    ASI_MASTER_GET_LPS_CNF_DATA_T      tData;    /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LPS_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		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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005225	ASI_MASTER_GET_LPS_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_LPS_CNF_DATA_T			
	abProjectedList[ASI_MASTER_LIST_SIZE]	UINT8[]		List of the projected slaves

Table 100: ASI_MASTER_PACKET_GET_LPS_CNF_T – Confirmation for Get List of projected Slaves Request

6.1.19 ASI_MASTER_GET_LAS_REQ/CNF – Get List of Activated Slaves

This packet can be used to obtain a list of slaves which have been activated.

This List of Activated Slaves (LAS) is stored in variable `abActivatedList`. It consists of 8 bytes. These are represented as a bit field for the corresponding channel. The following table shows, which bit of the LAS is related to which slave.

The LAS is part of the Extended Status Block, see there.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Byte #								
1	7	6	5	4	3	2	1	0
2	15	14	13	12	11	10	9	8
3	23	22	21	20	19	18	17	16
...								
8	x	62	61	60	59	58	57	56

Table 101: Representation of List of Activated Slaves (LAS)

If the bit of the corresponding slave is:

- set - slave is activated,
- not set - slave is not activated.

X means the value does not matter.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_GET_LAS_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LAS_REQ_Ttag
    ASI_MASTER_PACKET_GET_LAS_REQ_T;

struct ASI_MASTER_PACKET_GET_LAS_REQ_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LAS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005226	ASI_MASTER_GET_LAS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 102: ASI_MASTER_PACKET_GET_LAS_REQ_T – Get List of Activated Slaves Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_LAS_CNF_DATA_Ttag</code> */
#define ASI_MASTER_LIST_SIZE 8

typedef struct ASI_MASTER_GET_LAS_CNF_DATA_Ttag
    ASI_MASTER_GET_LAS_CNF_DATA_T;

struct ASI_MASTER_GET_LAS_CNF_DATA_Ttag
{
    TLR_UINT8    abActivatedList[ASI_MASTER_LIST_SIZE];
};

/** type of <code>ASI_MASTER_PACKET_GET_LAS_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LAS_CNF_Ttag
    ASI_MASTER_PACKET_GET_LAS_CNF_T;

struct ASI_MASTER_PACKET_GET_LAS_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
    ASI_MASTER_GET_LAS_CNF_DATA_T      tData;    /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LAS_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		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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005227	ASI_MASTER_GET_LAS_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_LAS_CNF_DATA_T			
	abActivatedList[ASI_MASTER_LIST_SIZE]	UINT8[]		List of the activated slaves

Table 103: ASI_MASTER_PACKET_GET_LAS_CNF_T –Confirmation for Get List of Activated Slaves Request

6.1.20 ASI_MASTER_GET_LDS_REQ/CNF – Get List of Detected Slaves

This packet can be used to obtain a list of slaves which have been detected.

This List of Detected Slaves (LDS) is stored in variable `abDetectedList`. It consists of 8 bytes. These are represented as a bit field for the corresponding channel. The following table shows, which bit of the LDS is related to which slave.

The LDS is part of the Extended Status Block, see there.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Byte #								
1	7	6	5	4	3	2	1	0
2	15	14	13	12	11	10	9	8
3	23	22	21	20	19	18	17	16
...								
8	x	62	61	60	59	58	57	56

Table 104: Representation of List of Detected Slaves (LDS)

If the bit of the corresponding slave is:

- set - slave is detected,
- not set - slave is not detected.

X means the value does not matter.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_GET_LDS_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LDS_REQ_Ttag
    ASI_MASTER_PACKET_GET_LDS_REQ_T;

struct ASI_MASTER_PACKET_GET_LDS_REQ_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LDS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005228	ASI_MASTER_GET_LDS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 105: ASI_MASTER_PACKET_GET_LDS_REQ_T – Get List of Detected Slaves Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_LDS_CNF_DATA_Ttag</code> */
#define ASI_MASTER_LIST_SIZE 8

typedef struct ASI_MASTER_GET_LDS_CNF_DATA_Ttag
    ASI_MASTER_GET_LDS_CNF_DATA_T;

struct ASI_MASTER_GET_LDS_CNF_DATA_Ttag
{
    TLR_UINT8    abDetectedList[ASI_MASTER_LIST_SIZE];
};
/** type of <code>ASI_MASTER_PACKET_GET_LDS_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LDS_CNF_Ttag
    ASI_MASTER_PACKET_GET_LDS_CNF_T;

struct ASI_MASTER_PACKET_GET_LDS_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
    ASI_MASTER_GET_LDS_CNF_DATA_T      tData;    /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LDS_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		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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005229	ASI_MASTER_GET_LDS_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_LDS_CNF_DATA_T			
	abDetectedList[ASI_MASTER_LIST_SIZE]	UINT8[]		List of the detected slaves

Table 106: ASI_MASTER_PACKET_GET_LDS_CNF_T –Confirmation for Get List of Detected Slaves Request

6.1.21 ASI_MASTER_GET_LPF_REQ/CNF – Get List of Peripheral Faults

This packet can be used to obtain a list of slaves with peripheral faults.

This List of Peripheral Faults (LPF) is stored in variable `abPeripheryFaultList`. It consists of 8 bytes. These are represented as a bit field for the corresponding channel. The following table shows, which bit of the LPF is related to which slave.

The LPF is part of the Extended Status Block, see there.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Byte #								
1	7	6	5	4	3	2	1	0
2	15	14	13	12	11	10	9	8
3	23	22	21	20	19	18	17	16
...								
8	x	62	61	60	59	58	57	56

Table 107: Representation of List of Peripheral Faults (LPF)

If the bit of the corresponding slave is:

- set - slave reports periphery fault,
- not set - slave reports periphery fault.

X means the value does not matter.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_GET_LPF_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LPF_REQ_Ttag
    ASI_MASTER_PACKET_GET_LPF_REQ_T;

struct ASI_MASTER_PACKET_GET_LPF_REQ_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LPF_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000522A	ASI_MASTER_GET_LPF_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 108: ASI_MASTER_PACKET_GET_LPF_REQ_T – Get List of Peripheral Faults Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_GET_LPF_CNF_DATA_Ttag</code> */
#define ASI_MASTER_LIST_SIZE 8

typedef struct ASI_MASTER_GET_LPF_CNF_DATA_Ttag
    ASI_MASTER_GET_LPF_CNF_DATA_T;

struct ASI_MASTER_GET_LPF_CNF_DATA_Ttag
{
    TLR_UINT8    abPeripheryFaultList[ASI_MASTER_LIST_SIZE];
};
/** type of <code>ASI_MASTER_PACKET_GET_LPF_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_LPF_CNF_Ttag
    ASI_MASTER_PACKET_GET_LPF_CNF_T;

struct ASI_MASTER_PACKET_GET_LPF_CNF_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
    ASI_MASTER_GET_LPF_CNF_DATA_T      tData;    /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_LPF_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		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	8	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000522B	ASI_MASTER_GET_LPF_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	
tData	structure ASI_MASTER_GET_LPF_CNF_DATA_T			
	abPeripheryFaultList[ASI_MASTER_LIST_SIZE]	UINT8[]		List of Peripheral Faults

Table 109: ASI_MASTER_PACKET_GET_LPF_CNF_T –Confirmation for Get List of Peripheral Faults Request

6.1.22 ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_REQ/CNF – Get Execution Control Flags

This packet is used to obtain the execution control flags (ASI flags).

These flags signal the state of the AS-Interface Master for the related channel to the user application and allows the application to control the Master.

The ASI flags word `usAsiFlags` is a bit field containing the following information:

Position	Flag	Explanation
Bit 0	ASI_MASTER_ASI_FLAGS_CONFIG_OK	CONFIGURATION OK: This flag signals that the configuration is correct.
Bit 1	ASI_MASTER_ASI_FLAGS_LDS0	LDS0 (Slave with Zero Address Detected): This flag signals that a slave with zero address has been detected. If set, a slave addressed with zero is detected. LDS.0 represents the accompanying bit of the List of Detected Slaves (LDS).
Bit 2	ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_ASSIGN	AUTO_ADDRESS_ASSIGN: This flag signals that automatic address assignment is possible. If set, Automatic Address Assignment is possible.
Bit 3	ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_AVAILABLE	AUTO_ADDRESS_AVAILABLE: This flag signals that automatic address assignment is possible. If set, Automatic Address Assignment will be processed as soon as a slave with addressed with zero and a valid configuration data occurred.
Bit 4	ASI_MASTER_ASI_FLAGS_CONFIGURATION_ACTIVE	CONFIGURATION_ACTIVE: This flag signals that the configuration mode active. If set, the Master is in configuration mode.
Bit 5	ASI_MASTER_ASI_FLAGS_NORMAL_OPERATION_ACTIVE	NORMAL_OPERATION_ACTIVE: This flag signals that normal operation is active. If set, the Master is in normal operation mode.

Bit 6	ASI_MASTER_ASI_FLAGS_APF_NOT_APO	APF_NOT_APO: This flag signals an ASi power failure. If set, AS-Interface system power is low or power down occurred during data transmission.
Bit 7	ASI_MASTER_ASI_FLAGS_OFFLINE_READY	OFFLINE_READY: Offline ready If set, the Master is in offline phase.
Bit 8	ASI_MASTER_ASI_FLAGS_PERIPHERY_OK	PERIPHERY OK: This flag signals that no peripheral fault has been detected. This function is not supported yet. The current state of this flag is 'not set'.

Table 110: Meaning of ASI Flags

Bits 9 to 15 are currently not used and reserved for later use.

Packet Structure Reference

```

/*****
typedef struct ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_REQ_Ttag
    ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_REQ_T;

struct ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_REQ_Ttag
{
    TLR_PACKET_HEADER_T                                tHead;/** packet header. */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_REQ_T					
Type: Request					
Area	Variable	Type	Value / Range	Description	
tHead	structure TLR_PACKET_HEADER_T				
		ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
		ulCmd	UINT32	0x0000522C	ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_REQ - Command
		ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
		ulRout	UINT32	x	Routing, do not touch

Table 111: ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_REQ_T – Get Execution Control Flags Request

Packet Structure Reference

```

/*****
#define ASI_MASTER_ASI_FLAGS_CONFIG_OK                0x0001
#define ASI_MASTER_ASI_FLAGS_LDS0                    0x0002
#define ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_ASSIGN     0x0004
#define ASI_MASTER_ASI_FLAGS_AUTO_ADDRESS_AVAILABLE 0x0008
#define ASI_MASTER_ASI_FLAGS_CONFIGURATION_ACTIVE    0x0010
#define ASI_MASTER_ASI_FLAGS_NORMAL_OPERATION_ACTIVE 0x0020
#define ASI_MASTER_ASI_FLAGS_APF_NOT_APO            0x0040
#define ASI_MASTER_ASI_FLAGS_OFFLINE_READY          0x0080
#define ASI_MASTER_ASI_FLAGS_PERIPHERY_OK           0x0100

typedef struct ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_CNF_DATA_Ttag
    ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_CNF_DATA_T;

struct ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_CNF_DATA_Ttag
{
    TLR_UINT16 usAsiFlags;
};

typedef struct ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_CNF_Ttag
    ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_CNF_T;

struct ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_CNF_DATA_T  tData;  /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_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		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	2	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000522D	ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_EXECUTION_CONTROL_FLAGS_CNF_DATA_T			
	usAsiFlags	UINT16		ASI Flags

Table 112: ASI_MASTER_PACKET_GET_EXECUTION_CONTROL_FLAGS_CNF_T – Confirmation for Get Execution Control Flags Request

6.1.23 ASI_MASTER_READ_IDENTIFICATION_STRING_REQ/CNF – Read Identification String

This service has to be used by the AP-Task in order to request the ASIMASTER-Task to read the identification string of an activated Slave with profile S-7.4 via combined transaction type 1. The ASIMASTER-Task will stop the cyclic process data exchange with the requested slave and continue with cyclic process data exchange, after the service is complete.

The `ulAddress` parameter defines the address of the requested slave. Slaves in extended addressing mode do not support combined transaction type 1, therefore the maximum value for `ulAddress` is limited to 31 for this service.

The maximum number of data is limited to 219 bytes in this implementation.

The identification string is explained in *Figure 7: Identification String (Read)* on page 178 and within the subsequent tables.

The complete identification string is described individually in the documentation of the corresponding Slave.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER-Task process queue.

Identification String

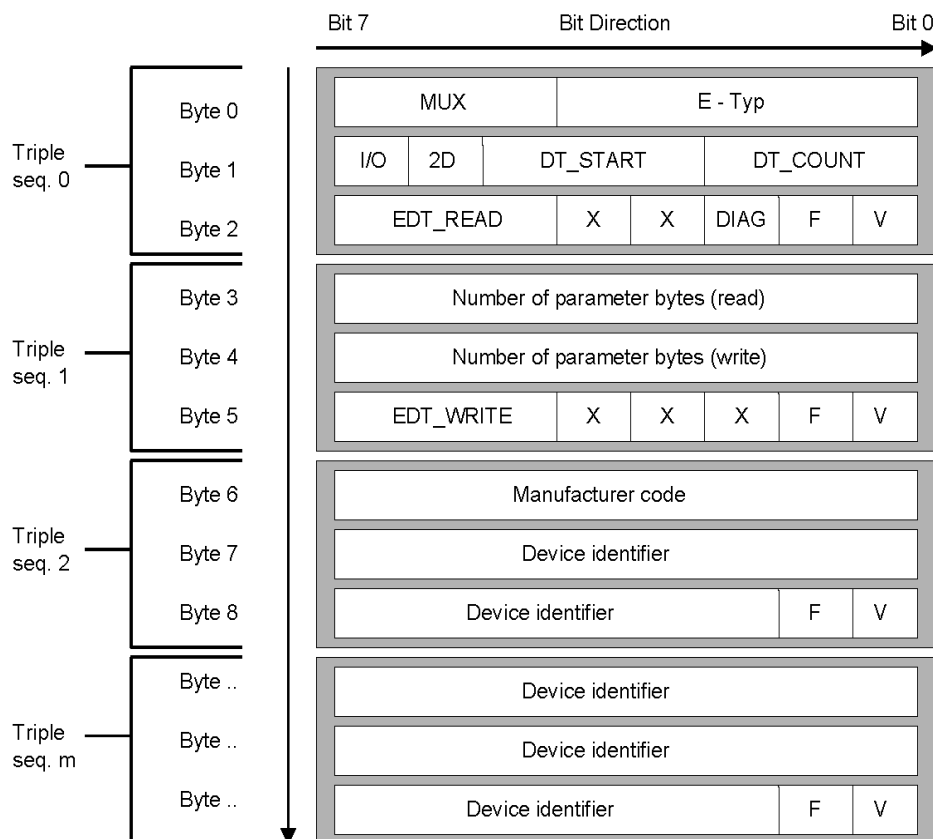


Figure 7: Identification String (Read)

The elements of the identification string are defined as follows:

Note: The complete identification string should however be described individually within the documentation of the corresponding Slave.

ID String Triple 1, Byte 0

Bits	Elements	Description
7-5	MUX	The element MUX shows the number of multiplexed channels of the slave <ul style="list-style-type: none"> • 0 = 1 channel, • 1 = 2 channels, • ..., • 7 = 8 channels
4-0	E-TYPE	This element specifies the slave type in relation to its functional behavior: <ul style="list-style-type: none"> • 0 = E3, E2, E1 are not used • 1 = multiplex representation of data • 2 = 16 bit digital values (S-7.4), reserved (else) • 3 = 4I / 4O operation • 4 .. 31: reserved

Table 113: Contents of ID String Triple 1, Byte 0

ID String Triple 1, Byte 1

Bits	Elements	Description
7	I/O	Indicates whether the slave has outputs or inputs during data transfer
6	2D	Indicates whether or not double data transfer is supported
5-3	DT_START	Indicates with which triple the normal data has to begin
2-0	DT_COUNT	Indicates which quantity of triples can be transferred during normal data transfer

Table 114: Contents of ID String Triple 1, Byte 1

ID String Triple 1, Byte 2

Bits	Elements	Description
7-5	EDT_READ	Reserved for extended data transfer, not defined yet, always set to 0
4-3	X	Unused, always set to 0
2	DIAG	Indicates whether or not the slave offers the possibility of reading the diagnostic string
1	F	Follow bit
0	V	Valid bit

Table 115: Contents of ID String Triple 1, Byte 2

ID String Triple 2, Byte 3

Bits	Elements	Description
7-0	Number of parameter bytes read	Specifies the number of parameter bytes the Master is able to read from the slave

Table 116: Contents of ID String Triple 2, Byte 3

ID String Triple 2, Byte 4

Bits	Elements	Description
7-0	Number of parameter bytes write	Specifies the number of parameter bytes the Master is capable of writing to the slave

Table 117: Contents of ID String Triple 2, Byte 4

ID String Triple 2, Byte 5

Bits	Elements	Description
7-5	EDT_WRITE	Reserved for extended data transfer, not defined yet, always set to 0
4-2	X	Unused, always set to 0
1	F	Follow bit
0	V	Valid bit

Table 118: Contents of ID String Triple 2, Byte 5

ID String Triple 3, Byte 6

Bits	Elements	Description
7-0	Manufacturer code	This code number is assigned to members of AS-Interface Association and is specified on the slave's data sheet.

Table 119: Contents of ID String Triple 3, Byte 6

ID String Triple 3, Byte 7

Bits	Elements	Description
7-0	Device identifier	This code is assigned by the manufacturer and is specified in the slave's data sheet.

Table 120: Contents of ID String Triple 3, Byte 7

ID String Triple 3, Byte 8

Bits	Elements	Description
7-2	Device identifier	This code is assigned by the manufacturer and is specified in the AS-Interface Slave's data sheet.
1	F	Follow bit
0	V	Valid bit

Table 121: Contents of ID String Triple 3, Byte 8

For each further slave, this triple of bits can be repeated.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_READ_IDENTIFICATION_STRING_REQ_DATA_Ttag</code> */
typedef struct ASI_MASTER_READ_IDENTIFICATION_STRING_REQ_DATA_Ttag
    ASI_MASTER_READ_IDENTIFICATION_STRING_REQ_DATA_T;

struct ASI_MASTER_READ_IDENTIFICATION_STRING_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};

/** type of <code>ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_Ttag
    ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_T;

struct ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_READ_IDENTIFICATION_STRING_REQ_DATA_T  tData;  /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005230	ASI_MASTER_READ_IDENTIFICATION_STRING_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_READ_IDENTIFICATION_STRING_REQ_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose identification string is requested

Table 122: ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_REQ_T – Read Identification String Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_READ_IDENTIFICATION_STRING_CNF_DATA_Ttag</code> */
typedef struct ASI_MASTER_READ_IDENTIFICATION_STRING_CNF_DATA_Ttag
    ASI_MASTER_READ_IDENTIFICATION_STRING_CNF_DATA_T;

#define ASI_MASTER_MAX_READ_IDENTIFICATION_STRING_DATA 219

struct ASI_MASTER_READ_IDENTIFICATION_STRING_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT8  abData[ASI_MASTER_MAX_READ_IDENTIFICATION_STRING_DATA];
};

/** type of <code>ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_CNF_Ttag
    ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_CNF_T;

struct ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_READ_IDENTIFICATION_STRING_CNF_DATA_T  tData;  /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_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		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	223	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005231	ASI_MASTER_READ_IDENTIFICATION_STRING_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_READ_IDENTIFICATION_STRING_CNF_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose identification string is requested
	abData[ASI_MASTER_MAX_READ_IDENTIFICATION_STRING_DATA]	UINT8[]		Area containing the requested identification string

Table 123: ASI_MASTER_PACKET_READ_IDENTIFICATION_STRING_CNF_T – Confirmation for Read Identification String Request

6.1.24 ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ/CNF – Read Diagnostic String

This service has to be used by the AP-Task in order to request the ASIMASTER-Task to read the diagnostic string of an activated (analog) slave with profile S-7.4 via combined transaction type 1. The ASIMASTER-Task will stop the cyclic process data exchange with the requested slave and continue with cyclic process data exchange, after the service is complete.

The `ulAddress` parameter defines the address of the requested slave. Slaves in extended addressing mode do not support combined transaction type 1, therefore the maximum value for `ulAddress` is limited to 31 for this service.

The `abData` structure of the confirmation packet contains the diagnostic string after being read out from the slave.

The module's **Diagnostic String** is a byte string of multiples of 3 byte. This string should be described in the data sheet of the appropriate slave.

The Diagnostic String is structured as follows:

Diagnostic String Triple 1, Byte 0

Bits	Elements	Description
7-0	Diagnosis byte	Diagnosis byte, refer to data sheet of the slave

Table 124: Contents of Diagnostic String Triple 1, Byte 0

Diagnostic String Triple 1, Byte 1

Bits	Elements	Description
7-0	Diagnosis byte	Diagnosis byte, refer to data sheet of the slave

Table 125: Contents of Diagnostic String Triple 1, Byte 1

Diagnostic String Triple 1, Byte 2

Bits	Elements	Description
7-2	X	Unused, always 0
1	F	Follow bit
0	V	Valid bit

Table 126: Contents of Diagnostic String Triple 1, Byte 2

For each further slave, this triple of bits can be repeated.

The maximum number of data is limited to 220 bytes in this implementation.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER-Task process queue.

Packet Structure Reference

```
/*
*****
** type of <code>ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ_DATA_Ttag</code> */
typedef struct ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ_DATA_Ttag
    ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ_DATA_T;

struct ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};

** type of <code>ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_Ttag
    ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_T;

struct ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_Ttag
{
    TLR_PACKET_HEADER_T                tHead;    /** packet header. */
    ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ_DATA_T    tData;    /** packet data */
};
*****
*/
```

Packet Description

structure ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005232	ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_READ_DIAGNOSTIC_STRING_REQ_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose diagnostic string is requested

Table 127: ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_REQ_T – Read Diagnostic String Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF_DATA_Ttag</code> */
typedef struct ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF_DATA_Ttag
    ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF_DATA_T;

#define ASI_MASTER_MAX_READ_DIAGNOSTIC_STRING_DATA 220

struct ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT8  abData[ASI_MASTER_MAX_READ_DIAGNOSTIC_STRING_DATA];
};

/** type of <code>ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_CNF_Ttag
    ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_CNF_T;

struct ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF_DATA_T  tData;  /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_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		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	334	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005233	ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_READ_DIAGNOSTIC_STRING_CNF_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose diagnostic string is requested
	abData[ASI_MASTER_MAX_READ_DIAGNOSTIC_STRING_DATA]	UINT8[]		Area containing the requested diagnostic data string

Table 128: ASI_MASTER_PACKET_READ_DIAGNOSTIC_STRING_CNF_T –Confirmation for Read Diagnostic String Request

6.1.25 ASI_MASTER_READ_PARAMETER_STRING_REQ/CNF – Read Parameter String

This service has to be used by the AP-Task in order to request the ASIMASTER-Task to read the parameter string of an activated slave with profile S-7.4 via combined transaction type 1. The ASIMASTER-Task will stop the cyclic process data exchange with the requested slave and continue with cyclic process data exchange, after the service is complete.

The `ulAddress` parameter of the request packet defines the address of the slave to be read out. Slaves in extended addressing mode do not support combined transaction type 1, therefore the maximum value for `ulAddress` is limited to 31 for this service.

The `abData` structure of the confirmation packet contains the parameter string after being read out from the slave.

The module's **Parameter String** is a byte string of multiples of 3 byte. This string should be described in the data sheet of the appropriate slave.

The Parameter String is structured as follows:

Parameter String Triple 1, Byte 0

Bits	Elements	Description
7-0	Parameter byte	Parameter byte, refer to data sheet of the slave

Table 129: Contents of Diagnostic String Triple 1, Byte 0

Parameter String Triple 1, Byte 1

Bits	Elements	Description
7-0	Parameter byte	Parameter byte, refer to data sheet of the slave

Table 130: Contents of Diagnostic String Triple 1, Byte 1

Parameter String Triple 1, Byte 2

Bits	Elements	Description
7-2	X	Unused, always 0
1	F	Follow bit
0	V	Valid bit

Table 131: Contents of Diagnostic String Triple 1, Byte 2

For each further slave, this triple of bits can be repeated.

The maximum number of data is limited to 220 bytes in this implementation.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER-Task process queue.

Packet Structure Reference

```
/*
*****
** type of <code>ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_Ttag</code> */
typedef struct ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_Ttag
    ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_T;

struct ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};
** type of <code>ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_Ttag
    ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_T;

struct ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_T  tData;  /** packet data */
};
*****
*/
```

Packet Description

structure ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005234	ASI_MASTER_READ_PARAMETER_STRING_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose parameter string is requested

Table 132: ASI_MASTER_PACKET_READ_PARAMETER_STRING_REQ_T – Read Parameter String Request

Packet Structure Reference

```
/*
/*****
** type of <code>ASI_MASTER_READ_PARAMETER_STRING_CNF_DATA_Ttag</code> */
typedef struct ASI_MASTER_READ_PARAMETER_STRING_CNF_DATA_Ttag
    ASI_MASTER_READ_PARAMETER_STRING_CNF_DATA_T;

#define ASI_MASTER_MAX_READ_PARAMETER_STRING_DATA 220

struct ASI_MASTER_READ_PARAMETER_STRING_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT8  abData[ASI_MASTER_MAX_READ_PARAMETER_STRING_DATA];
};

/** type of <code>ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNF_Ttag
    ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNF_T;

struct ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_READ_PARAMETER_STRING_CNF_DATA_T  tData;  /** packet data */
};
/*****/
```

Packet Description

structure ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNFT				
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, 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	334	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005235	ASI_MASTER_READ_PARAMETER_STRING_CNFT - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_READ_PARAMETER_STRING_CNFT_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose parameter string is requested
	abData[ASI_MASTER_MAX_READ_PARAMETER_STRING_DATA]	UINT8[]		Area containing the requested parameter string

Table 133: ASI_MASTER_PACKET_READ_PARAMETER_STRING_CNFT – Confirmation for Read Parameter String Request

6.1.26 ASI_MASTER_WRITE_PARAMETER_STRING_REQ/CNF – Write Parameter String

This service has to be used by the AP-Task in order to request the ASIMASTER -Task to write the parameter string to an activated slave with profile S-7.4 via combined transaction type 1. The ASIMASTER -Task will stop the cyclic process data exchange with the requested Slave and continue with cyclic process data exchange, after the service is complete.

The `ulAddress` parameter defines the address of the requested slave. Slaves in extended addressing mode do not support combined transaction type 1, therefore the maximum value for `ulAddress` is limited to 31 for this service.

The `abData` structure of the request packet defines the parameter string to be written to the slave. The contents of `abData` has to be structured just as described in the preceding section ASI_MASTER_READ_PARAMETER_STRING_REQ/CNF – Read Parameter String, see there for more information.

The maximum number of data is limited to 220 bytes in this implementation.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER -Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_WRITE_PARAMETER_STRING_REQ_DATA_Ttag</code> */
typedef struct ASI_MASTER_WRITE_PARAMETER_STRING_REQ_DATA_Ttag
    ASI_MASTER_WRITE_PARAMETER_STRING_REQ_DATA_T;

#define ASI_MASTER_MAX_WRITE_PARAMETER_STRING_DATA 220

struct ASI_MASTER_WRITE_PARAMETER_STRING_REQ_DATA_Ttag
{
    TLR_UINT32 ulAddress;
    TLR_UINT8  abData[ASI_MASTER_MAX_WRITE_PARAMETER_STRING_DATA];
};

/** type of <code>ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_Ttag
    ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_T;

struct ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_WRITE_PARAMETER_STRING_REQ_DATA_T  tData;  /** packet data */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	334	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005236	ASI_MASTER_WRITE_PARAMETER_STRING_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_READ_PARAMETER_STRING_REQ_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose parameter string is written
	abData[ASI_MASTER_MAX_READ_PARAMETER_STRING_DATA]	UINT8[]		Area containing the parameter string to be written

Table 134: ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_REQ_T – Write Parameter String Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_WRITE_PARAMETER_STRING_CNF_DATA_Ttag</code> */
typedef struct ASI_MASTER_WRITE_PARAMETER_STRING_CNF_DATA_Ttag
    ASI_MASTER_WRITE_PARAMETER_STRING_CNF_DATA_T;

struct ASI_MASTER_WRITE_PARAMETER_STRING_CNF_DATA_Ttag
{
    TLR_UINT32 ulAddress;
};
/** type of <code>ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_CNF_Ttag
    ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_CNF_T;

struct ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_WRITE_PARAMETER_STRING_CNF_DATA_T tData; /** packet data */
};
*****/
    
```

Packet Description

structure ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_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		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	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005237	ASI_MASTER_WRITE_PARAMETER_STRING_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	
tData	structure ASI_MASTER_READ_PARAMETER_STRING_CNF_DATA_T			
	ulAddress	UINT32	1 ... 31	Address of slave whose parameter string is written

Table 135: ASI_MASTER_PACKET_WRITE_PARAMETER_STRING_CNF_T –Confirmation for Write Parameter String Request

6.1.27 ASI_MASTER_SET_OPERATION_MODE_REQ/CNF – Set Operation Mode

This service has to be used by the AP-Task in order to request the ASIMASTER task to set the operation mode to protected- or configuration mode.

- In the protected mode, the AS-Interface Master activates only projected slaves. Additionally, the projected input/output configuration and all identification codes must match to the connected slave.
- In configuration mode, all detected slaves will be activated by the AS-Interface Master.

The `ulOperationMode` parameter defines whether the AS-Interface Master should change to the protected- (`ASI_MASTER_OPERATION_MODE_PROTECTED`) or configuration mode (`ASI_MASTER_OPERATION_MODE_CONFIGURATION`).



Note: If a slave with zero address is detected, the operation mode cannot be changed to protected mode.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_SET_OPERATION_MODE_REQ_DATA_T</code> */
#define ASI_MASTER_OPERATION_MODE_CONFIGURATION      0x00000000L
#define ASI_MASTER_OPERATION_MODE_PROTECTED        0x00000001L

typedef struct ASI_MASTER_SET_OPERATION_MODE_REQ_DATA_Ttag
  ASI_MASTER_SET_OPERATION_MODE_REQ_DATA_T;

struct ASI_MASTER_SET_OPERATION_MODE_REQ_DATA_Ttag
{
  TLR_UINT32 ulOperationMode;
};

/** type of <code>ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_Ttag
  ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_T;

struct ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_Ttag
{
  TLR_PACKET_HEADER_T          tHead;  /** packet header.  */
  ASI_MASTER_SET_OPERATION_MODE_REQ_DATA_T tData;  /** packet data    */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005238	ASI_MASTER_SET_OPERATION_MODE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_SET_OPERATION_MODE_REQ_DATA_T			
	ulOperationMode	UINT32	0,1	Flag that indicates if set to ASI_MASTER_OPERATION_MODE_CONFIGURATION (0) that the AS-Interface Master should change to the configuration mode. If set to ASI_MASTER_OPERATION_MODE_PROTECTED (1), the AS-Interface Master should change to the protected mode.

Table 136: ASI_MASTER_PACKET_SET_OPERATION_MODE_REQ_T – Set Operation Mode Request

Packet Structure Reference

```

/** type of <code>ASI_MASTER_PACKET_SET_OPERATION_MODE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_OPERATION_MODE_CNF_Ttag
    ASI_MASTER_PACKET_SET_OPERATION_MODE_CNF_T;

struct ASI_MASTER_PACKET_SET_OPERATION_MODE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header. */
};

```

Packet Description

structure ASI_MASTER_PACKET_SET_OPERATION_MODE_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		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005239	ASI_MASTER_SET_OPERATION_MODE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	

Table 137: ASI_MASTER_PACKET_SET_OPERATION_MODE_CNF_T –Confirmation for Set Operation Mode Request

6.1.28 ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ/CNF – Set Data Exchange Active

This service has to be used by the AP-Task in order to request the ASIMASTER task to set data exchange to active or inactive state.

- If data exchange is set to active state, the AS-Interface Master cyclically sends data exchange requests to all activated Slaves.
- If data exchange is set to inactive state, no data exchange will be performed, instead the AS-Interface Master cyclically reads the identification codes from all activated Slaves.

The `ulDataExchangeActive` parameter defines whether the AS-Interface Master should set the data exchange to active or inactive state. Active state is possible either online or offline.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ_DATA_T</code> */
typedef struct ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ_DATA_Ttag
    ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ_DATA_T;

#define ASI_MASTER_DATA_EXCHANGE_INACTIVE          0x00000000L
#define ASI_MASTER_DATA_EXCHANGE_ACTIVE          0x00000001L
#define ASI_MASTER_DATA_EXCHANGE_OFFLINE_ACTIVE  0x00000002L

struct ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ_DATA_Ttag
{
    TLR_UINT32 ulDataExchangeActive;
};

/** type of <code>ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_Ttag
    ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_T;

struct ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ_DATA_T tData; /** packet data */
};
/*****

```

Packet Description

structure ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000523A	ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_REQ_DATA_T			
	ulDataExchangeActive	UINT32	0-2	Flag that indicates if set to ASI_MASTER_DATA_EXCHANGE_ACTIVE (1) that the AS-Interface Master should set data exchange to active state. If set to ASI_MASTER_DATA_EXCHANGE_INACTIVE (0), the AS-Interface Master should set data exchange to inactive state. If set to ASI_MASTER_DATA_EXCHANGE_OFFLINE_ACTIVE (2), the AS-Interface Master should set data exchange to active state and offline mode.

Table 138: ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_REQ_T – Set Data Exchange Active Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_CNF_Ttag
    ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_CNF_T;

struct ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header. */
};
/*****
    
```

Packet Description

structure ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_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		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000523B	ASI_MASTER_SET_DATA_EXCHANGE_ACTIVE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	

Table 139: ASI_MASTER_PACKET_SET_DATA_EXCHANGE_ACTIVE_CNF_T – Confirmation for Set Data Exchange Active Request

6.1.29 ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ/CNF – Set Auto Address Enable

This service has to be used by the AP-Task in order to request the ASIMASTER task to enable or to disable the auto addressing.

If the master detects a slave which can replace a missing slave (it needs the same I/O Code, ID, ID1 and ID2 Code) it assigns the address of the missing slave to the detected one, if the new slave has zero address and auto addressing is enabled. Auto addressing can only be processed if one, and only one projected slave is missing.

The `ulAutoAddress` parameter defines whether the AS-Interface Master should enable (`ASI_MASTER_AUTO_ADDRESS_ENABLE = 1`) or disable (`ASI_MASTER_AUTO_ADDRESS_DISABLE = 0`) the auto addressing service.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ_DATA_T</code> */
#define ASI_MASTER_AUTO_ADDRESS_DISABLE          0x00000000L
#define ASI_MASTER_AUTO_ADDRESS_ENABLE          0x00000001L

typedef struct ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ_DATA_Ttag
    ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ_DATA_T;

struct ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ_DATA_Ttag
{
    TLR_UINT32 ulAutoAddress;
};

/** type of <code>ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_Ttag
    ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_T;

struct ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_Ttag
{
    TLR_PACKET_HEADER_T          tHead;    /** packet header. */
    ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ_DATA_T  tData;    /** packet data */
};
/*****/

```

Packet Description

structure ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000523C	ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_REQ_DATA_T			
	ulAutoAddress	UINT32	0,1	Flag that indicates if set to ASI_MASTER_AUTO_ADDRESS_ENABLE (1) that the AS-Interface Master should enable the auto addressing mode. If set to ASI_MASTER_AUTO_ADDRESS_DISABLE (0), the AS-Interface Master should disable the auto addressing mode.

Table 140: ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_REQ_T – Set Auto Address Enable Request

Packet Structure Reference

```

/** type of <code>ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_CNF_Ttag
    ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_CNF_T;

struct ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header. */
};

```

Packet Description

structure ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_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		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000523D	ASI_MASTER_SET_AUTO_ADDRESS_ENABLE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	

Table 141: ASI_MASTER_PACKET_SET_AUTO_ADDRESS_ENABLE_CNF_T – Confirmation for Set Auto Address Enable Request

6.1.30 ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_REQ/CNF – Get Auto Address Enable

This service has to be used by the AP-Task in order to request the information from the ASIMASTER task whether auto addressing is enabled or to disabled.

The `ulAutoAddress` parameter indicates whether the auto addressing service of the AS-Interface Master is enabled or disabled.

The macro `TLR_QUE_SEND_PACKET_FIFO()` has to be used to send the packet to the ASIMASTER - Task process queue.

Packet Structure Reference

```

/*****
/** type of <code>ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_Ttag
    ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_T;

struct ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header. */
};
/****
    
```

Packet Description

structure ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIMASTER	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000523E	ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 142: ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_REQ_T – Get Auto Address Enable Request

Packet Structure Reference

```
/*
** type of <code>ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF_DATA_T</code> */
typedef struct ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF_DATA_Ttag
    ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF_DATA_T;

struct ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF_DATA_Ttag
{
    TLR_UINT32 ulAutoAddress;
};

/*
** type of <code>ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_CNF_Ttag</code> */
typedef struct ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_CNF_Ttag
    ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_CNF_T;

struct ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;  /** packet header. */
    ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF_DATA_T tData; /** packet data */
};
*/
```

Packet Description

structure ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_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		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	4	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x0000523F	ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change
tData	structure ASI_MASTER_GET_AUTO_ADDRESS_ENABLE_CNF_DATA_T			
	ulAutoAddress	UINT32	0,1	Flag indicating if set to ASI_MASTER_AUTO_ADDRESS_ENABLE (1) that the The auto addressing mode of AS-Interface Master mode is enabled. If set to ASI_MASTER_AUTO_ADDRESS_DISABLE (0), the The auto addressing mode of AS-Interface Master mode is disabled.

Table 143: ASI_MASTER_PACKET_GET_AUTO_ADDRESS_ENABLE_CNF_T – Confirmation for Get Auto Address Enable Request

6.1.31 ASI_MASTER_STATE_CHANGE_IND/RES – State Change Indication

This indication informs about changes of state. It covers both the AS-Interface Master's state ([common status](#)) (see section *Common Status*). and the [extended status](#) (see section *Extended Status*).



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

```

/*****
/** type of <code>ASI_MASTER_STATE_CHANGE_IND_DATA_Ttag</code> */
typedef struct ASI_MASTER_STATE_CHANGE_IND_DATA_Ttag
    ASI_MASTER_STATE_CHANGE_IND_DATA_T;

struct ASI_MASTER_STATE_CHANGE_IND_DATA_Ttag
{
    TLR_UINT32                ulMasterState;
    NETX_MASTER_STATUS        tMasterState;
    ASI_MASTER_EXTENDED_STATE_T tExtendedState;
};

/** type of <code>ASI_MASTER_PACKET_STATE_CHANGE_IND_Ttag</code> */
typedef struct ASI_MASTER_PACKET_STATE_CHANGE_IND_Ttag
    ASI_MASTER_PACKET_STATE_CHANGE_IND_T;

struct ASI_MASTER_PACKET_STATE_CHANGE_IND_Ttag
{
    TLR_PACKET_HEADER_T        tHead;    /** packet header.          */
    ASI_MASTER_STATE_CHANGE_IND_DATA_T tData; /** packet request data.    */
};
*****/

```

Packet Description

structure ASI_MASTER_PACKET_STATE_CHANGE_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		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	124	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x000052FE	ASI_MASTER_STATE_CHANGE_IND - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_MASTER_STATE_CHANGE_IND_DATA_T			
	ulMasterState	UINT32		Master state value
	tMasterState	NETX_MASTER_STATUS		Structure for master state
	tExtendedState	ASI_MASTER_EXTENDED_STATE_T		Structure for extended state

Table 144: ASI_MASTER_PACKET_STATE_CHANGE_IND_T – State Change Indication

Packet Structure Reference

```

/** type of <code>ASI_MASTER_PACKET_STATE_CHANGE_RES_Ttag</code> */
typedef struct ASI_MASTER_PACKET_STATE_CHANGE_RES_Ttag
    ASI_MASTER_PACKET_STATE_CHANGE_RES_T;

struct ASI_MASTER_PACKET_STATE_CHANGE_RES_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header.          */
};

```

Packet Description

structure ASI_MASTER_PACKET_STATE_CHANGE_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	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet	
		ulSrcId	UINT32	Source End Point Identifier, specifying the origin of the packet inside the Source Process	
		ulLen	UINT32	0	Packet Data Length in bytes
		ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
		ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
		ulCmd	UINT32	0x000052FF	ASI_MASTER_STATE_CHANGE_RES - Command
		ulExt	UINT32	0	Extension, reserved
		ulRout	UINT32	x	Routing information, do not change

Table 145: ASI_MASTER_PACKET_STATE_CHANGE_RES_T – Response to State Change Indication

6.2 The ASIAPM -Task

The ASIAPM -Task only offers packets regarding initialization. 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 ASIAPM -Task the Macro `TLR_QUE_IDENTIFY()` has to be used in conjunction with the following ASCII-queue name

ASCII queue name	Description
"QUE_ASIAPM"	Name of the ASI APM-Task process queue

Table 146: ASIAPM -Task process queue

The diagnostic and error messages which can be issued by the APM-Task are described in section *Status/Error codes APM-Task* of this document.

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

No. of section	Packets	Page
6.2.1	ASI_APM_GET_STATE_REQ/CNF – Get State	214
6.2.2	ASI_APM_STORE_ACTUAL_CONFIG_REQ/CNF – Store Actual Configuration	216
6.2.3	ASI_APM_STORE_ACTUAL_PARAM_REQ/CNF – Store Actual Parameters	219
6.2.4	ASI_APM_SET_LPS_REQ/CNF – Set List of Projected Slaves	221

Table 147: Topics of ASIAPM -Task and associated packets

6.2.1 ASI_APM_GET_STATE_REQ/CNF – Get State

This packet allows to retrieve internal state information. It is reserved for future use.

Packet Structure Reference

```

/*****
/** type of <code>ASI_APM_PCK_GET_STATE_REQ_Ttag</code> */
typedef struct ASI_APM_PCK_GET_STATE_REQ_Ttag
    ASI_APM_PCK_GET_STATE_REQ_T;

struct ASI_APM_PCK_GET_STATE_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header */
};
/*****/

```

Packet Description

structure ASI_APM_PCK_GET_STATE_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIAPM	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005300L	ASI_APM_GET_STATE_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 148: ASI_APM_PCK_GET_STATE_REQ_T – Get State Request

Packet Structure Reference

```

/*****
/** type of <code>ASI_APM_GET_STATE_CNF_DATA_Ttag</code> */
typedef struct ASI_APM_GET_STATE_CNF_DATA_Ttag
    ASI_APM_GET_STATE_CNF_DATA_T;

struct ASI_APM_GET_STATE_CNF_DATA_Ttag
{
    TLR_UINT32 ulReserved;
};

/** type of <code>ASI_APM_PCK_GET_STATE_CNF_Ttag</code> */
typedef struct ASI_APM_PCK_GET_STATE_CNF_Ttag
    ASI_APM_PCK_GET_STATE_CNF_T;

struct ASI_APM_PCK_GET_STATE_CNF_Ttag
{
    TLR_PACKET_HEADER_T          tHead;    /** packet header */
    ASI_APM_GET_STATE_CNF_DATA_T tData;    /** packet data */
};
*****/
    
```

Packet Description

structure ASI_APM_PCK_GET_STATE_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		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	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005301	ASI_APM_GET_STATE_CNF - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch	
tData	structure			
	ulReserved	UINT32		Reserved for future use

Table 149: ASI_APM_PCK_GET_STATE_CNF_T –Confirmation for Get State Request

6.2.2 ASI_APM_STORE_ACTUAL_CONFIG_REQ/CNF – Store Actual Configuration

This packet can be sent from the host to the stack in order to request storing the current configuration.

In detail, the following actions are performed:

- the values of the configuration data image (CDI) are copied into the permanent configuration data (PCD)
- the values of the list of activated slaves (LAS) are copied into the list of projected slaves (LPS).
- For all slaves not included within the list of projected slaves (LPS) the permanent parameter data (PP) are set to the hexadecimal value 0xF

This function is restricted to configuration mode only. It is not applicable in protected mode. For more information about this topic refer to section *Operation Modes* on page 69 of this document.



Note: This packet corresponds to Execution control function #10 “Store Actual Configuration” defined in the Complete Specification of AS-Interface. See reference #3.

Packet Structure Reference

```
/** type of <code>ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_Ttag</code> */
typedef struct ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_Ttag
    ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_T;

struct ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header */
};
```

Packet Description

structure ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIAPM	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005302	ASI_APM_STORE_ACTUAL_CONFIG_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 150: ASI_APM_PCK_STORE_ACTUAL_CONFIG_REQ_T – Store Actual Configuration Request

Packet Structure Reference

```

/** type of <code>ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_Ttag</code> */
typedef struct ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_Ttag
    ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_T;

struct ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header */
};

```

Packet Description

structure ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	Description			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005303	ASI_APM_STORE_ACTUAL_CONFIG_CNF - Command
	ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change	

Table 151: ASI_APM_PCK_STORE_ACTUAL_CONFIG_CNF_T – Confirmation for Store Actual Configuration Request

6.2.3 ASI_APM_STORE_ACTUAL_PARAM_REQ/CNF – Store Actual Parameters

This packet can be sent from the host to the stack in order to request storing the current parameter set. The parameter values of the parameter image (PI) array are copied into the permanent parameter (PP) array. Here, the AS-Interface specification requires non-volatile storage of the data. Initially all bits of the permanent parameter array are set to the value „1“ for a factory new master.



Note: This packet corresponds to Execution control function #7 “Store Actual Parameters” defined in the Complete Specification of AS-Interface. See reference #3.

Packet Structure Reference

```
/** type of <code>ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_Ttag</code> */
typedef struct ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_Ttag
    ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_T;

struct ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_Ttag
{
    TLR_PACKET_HEADER_T tHead; /** packet header */
};
```

Packet Description

structure ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_T					
Type: Request					
Area	Variable	Type	Value / Range	Description	
tHead	structure TLR_PACKET_HEADER_T				
		ulDest	UINT32	0x20/ QUE_ASIAPM	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	0 ... 2 ³² -1	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 <i>Status/Error Codes Overview</i>
		ulCmd	UINT32	0x00005304L	ASI_APM_STORE_ACTUAL_PARAM_REQ - Command
		ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
		ulRout	UINT32	x	Routing, do not touch

Table 152: ASI_APM_PCK_STORE_ACTUAL_PARAM_REQ_T – Store Actual Parameters Request

Packet Structure Reference

```

/** type of <code>ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_Ttag</code> */
typedef struct ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_Ttag
    ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_T;

struct ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header */
};

```

Packet Description

structure ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	Description			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2 ³² -1	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005305L	ASI_APM_STORE_ACTUAL_PARAM_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change

Table 153: ASI_APM_PCK_STORE_ACTUAL_PARAM_CNF_T – Confirmation for Store Actual Parameters Request

6.2.4 ASI_APM_SET_LPS_REQ/CNF – Set List of Projected Slaves

This packet can be used to set a list of slaves which have been projected.

This List of Projected Slaves (LPS) is stored in variable `abLps`. It consists of 8 bytes. These are represented as a bit field for the corresponding channel. The following table shows, which bit of the LPS is related to which slave.

The LPS is part of the Extended Status Block, see there.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Byte #								
1	7	6	5	4	3	2	1	0
2	15	14	13	12	11	10	9	8
3	23	22	21	20	19	18	17	16
...								
8	x	62	61	60	59	58	57	56

Table 154: Representation of List of Projected Slaves (LPS)

If the bit of the corresponding slave is:

- set - slave is configured,
- not set - slave is not configured.

X means the value does not matter.

Note: This packet corresponds to Execution control function #12 Set_LPS defined in the Complete Specification of AS-Interface. See reference #3.

Packet Structure Reference

```

/** type of <code>ASI_APM_SET_LPS_REQ_DATA_Ttag</code> */
typedef struct ASI_APM_SET_LPS_REQ_DATA_Ttag
    ASI_APM_SET_LPS_REQ_DATA_T;

struct ASI_APM_SET_LPS_REQ_DATA_Ttag
{
    TLR_UINT8    abLps[ASI_MASTER_LIST_SIZE];
};

/** type of <code>ASI_APM_PCK_SET_LPS_REQ_Ttag</code> */
typedef struct ASI_APM_PCK_SET_LPS_REQ_Ttag
    ASI_APM_PCK_SET_LPS_REQ_T;

struct ASI_APM_PCK_SET_LPS_REQ_Ttag
{
    TLR_PACKET_HEADER_T    tHead; /** packet header */
    ASI_APM_SET_LPS_REQ_DATA_T tData; /** packet data */
};

```

Packet Description

structure ASI_APM_PCK_SET_LPS_REQ_T				
Type: Request				
Area	Variable	Type	Value / Range	Description
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ QUE_ASIAPM	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	0 ... $2^{32}-1$	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 <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005306	ASI_APM_SET_LPS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure ASI_APM_SET_LPS_REQ_DATA_T			
	abLps[ASI_MASTER_LIST_SIZE]	UINT8[]		List of Projected Slaves

Table 155: ASI_APM_PCK_SET_LPS_REQ_T – Set List of Projected Slaves Request

Packet Structure Reference

```

/** type of <code>ASI_APM_PCK_SET_LPS_CNF_Ttag</code> */
typedef struct ASI_APM_PCK_SET_LPS_CNF_Ttag
    ASI_APM_PCK_SET_LPS_CNF_T;

struct ASI_APM_PCK_SET_LPS_CNF_Ttag
{
    TLR_PACKET_HEADER_T tHead;    /** packet header */
};

```

Packet Description

structure ASI_APM_PCK_SET_LPS_CNF_T				
Type: Confirmation				
Area	Variable	Type	Value / Range	Description
tHead	Description			
	ulDest	UINT32		Destination queue handle, unchanged
	ulSrc	UINT32		Source queue handle, unchanged
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... $2^{32}-1$	Packet Identification, unchanged
	ulSta	UINT32		See section <i>Status/Error Codes Overview</i>
	ulCmd	UINT32	0x00005307	ASI_APM_SET_LPS_CNF - Command
	ulExt	UINT32	0	Extension, reserved
	ulRout	UINT32	x	Routing information, do not change

Table 156: ASI_APM_PCK_SET_LPS_CNF_T –Confirmation for Set List of Projected Slaves Request

7 Status/Error Codes Overview

7.1 Status/Error Codes ASIMASTER-Task

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_ASI_MASTER_PARAM_CYCLETIME (0xC07A0001L)	Invalid value for parameter cycle time.
TLR_E_ASI_MASTER_PARAM_XC_INSTANCE (0xC07A0002L)	Invalid value for parameter xC instance.
TLR_E_ASI_MASTER_PARAM_QUEUE_ELEMENT (0xC07A0003L)	Invalid value for parameter queue element.
TLR_E_ASI_MASTER_PARAM_POOL_ELEMENT (0xC07A0004L)	Invalid value for parameter pool element.
TLR_E_ASI_MASTER_PARAM_MIN_CYCLE_TIME (0xC07A0005L)	Invalid value for parameter min cycle timer.
TLR_E_ASI_MASTER_PARAM_OPERATION_MODE (0xC07A0006L)	Invalid value for parameter operation mode.
TLR_E_ASI_MASTER_PARAM_DATA_EXCHANGE (0xC07A0007L)	Invalid value for parameter data exchange.
TLR_E_ASI_MASTER_PARAM_AUTO_ADDRESS_ENABLE (0xC07A0008L)	Invalid value for parameter auto address enable.
TLR_E_ASI_MASTER_PARAM_MANAGEMENT_PHASE (0xC07A0009L)	Invalid value for parameter management phase.
TLR_E_ASI_MASTER_PARAM_PROCESS_DATA_MODE (0xC07A000AL)	Invalid value for parameter process data mode.
TLR_E_ASI_MASTER_PARAM_DATA_FORMAT (0xC07A000BL)	Invalid value for parameter process data format.
TLR_E_ASI_MASTER_INIT_BUFFER (0xC07A000CL)	Failed to initialize data buffer.
TLR_E_ASI_MASTER_INITIALIZING (0xC07A000DL)	Master is initializing.
TLR_E_ASI_MASTER_DATA_COUNT (0xC07A000EL)	Invalid data count.
TLR_E_ASI_MASTER_DATA_OFFSET (0xC07A000FL)	Invalid data offset.
TLR_E_ASI_MASTER_NOT_ALLOWED_IN_PROTECTED_MODE (0xC07A0010L)	Request is not allowed in protected mode.
TLR_E_ASI_MASTER_AUTO_CLEAR (0xC07A0011L)	Master is in auto-clear state.
TLR_E_ASI_MASTER_CONTROL_ERROR (0xC07A0012L)	Control error detected.
TLR_E_ASI_MASTER_SLAVE_MISSING (0xC07A0013L)	Slave is missing.
TLR_E_ASI_MASTER_POWER_FAILURE (0xC07A0014L)	Power failure detected.
TLR_E_ASI_MASTER_OFFLINE_READY (0xC07A0015L)	Master is in offline ready state.
TLR_E_ASI_MASTER_NOT_IN_NORMAL_OPERATION (0xC07A0016L)	Master is not in normal operation.
TLR_E_ASI_MASTER_INVALID_SLAVE_ADDRESS (0xC07A0017L)	Invalid slave address.
TLR_E_ASI_MASTER_SLAVE_ACTIVATED	Slave is activated.

Definition / (Value)	Description
(0xC07A0018L)	
TLR_E_ASI_MASTER_SLAVE_NOT_ACTIVATED (0xC07A0019L)	Slave is not activated.
TLR_E_ASI_MASTER_SLAVE_DETECTED (0xC07A001AL)	Slave is detected.
TLR_E_ASI_MASTER_SLAVE_NOT_DETECTED (0xC07A001BL)	Slave is not detected.
TLR_E_ASI_MASTER_TIMEOUT (0xC07A001CL)	Timeout detected.
TLR_E_ASI_MASTER_SLAVE_0_DETECTED (0xC07A001DL)	Slave at address 0 detected.
TLR_E_ASI_MASTER_NEW_SLAVE_DETECTED (0xC07A001EL)	Slave at new address detected.
TLR_E_ASI_MASTER_DELETE_ADDRESS (0xC07A001FL)	Error with deletion of address.
TLR_E_ASI_MASTER_READ_EXT_ID1 (0xC07A0020L)	Error with reading extended ID code 1.
TLR_E_ASI_MASTER_SET_EXT_ID1 (0xC07A0021L)	Error with setting extended ID code 1.
TLR_E_ASI_MASTER_ADDRESS_SET_TEMPORARY (0xC07A0022L)	New address stored temporarily.
TLR_E_ASI_MASTER_SET_ADDRESS (0xC07A0023L)	Error with setting new address.
TLR_E_ASI_MASTER_EXT_ID1_SET_TEMPORARY (0xC07A0024L)	Extended ID code 1 stored temporarily.
TLR_E_ASI_MASTER_INVALID_SLAVE_PROFILE (0xC07A0025L)	Invalid slave profile.
TLR_E_ASI_MASTER_SLAVE_CONFIG (0xC07A0026L)	Invalid slave configuration.
TLR_E_ASI_MASTER_SLAVE_ALREADY_CONFIGURED (0xC07A0027L)	Slave is already configured.
TLR_E_ASI_MASTER_STRING_TRANSFER_DATA_OVERFLOW (0xC07A0028L)	Data overflow during string transfer detected.
TLR_E_ASI_MASTER_PARAM_AUTOCLEAR_WITH_AUTOADDRESS (0xC07A0029L)	Invalid value for parameter autoclear in combination with value for parameter autoaddress.
TLR_E_ASI_MASTER_PARAM_AUTOCLEAR_WITH_CONFIG_MODE (0xC07A002AL)	Invalid value for parameter autoclear in combination with value for parameter operationmode.
TLR_E_ASI_MASTER_COMMAND_NOT_ALLOWED_WITH_AUTOCLEAR (0xC07A002BL)	Command is not allowed if autoclear is active.

Table 157: Status/Error Codes ASIMASTER -Task

7.2 Status/Error codes APM-Task

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_ASI_APM_PARAM_CYCLETIME (0xC07B0001L)	Invalid value for parameter cycle time.
TLR_E_ASI_APM_PARAM_CHN_INSTANCE (0xC07B0002L)	Invalid value for parameter channel instance.
TLR_E_ASI_APM_PARAM_QUEUE_ELEMENT (0xC07B0003L)	Invalid value for parameter queue element.
TLR_E_ASI_APM_PARAM_POOL_ELEMENT (0xC07B0004L)	Invalid value for parameter pool element.
TLR_E_ASI_APM_PARAM_AUTO_CLEAR (0xC07B0005L)	Invalid value for parameter auto-clear.
TLR_E_ASI_APM_SLAVE_ALREADY_CONFIGURED (0xC07B0006L)	Slave is already configured.
TLR_E_ASI_APM_INVALID_DBM_VERSION (0xC07B0007L)	Invalid version of database.
TLR_E_ASI_APM_STORE_CONFIGURATION_NOT_POSSIBLE (0xC07B0008L)	Permanent storage of configuration is not possible.
TLR_E_ASI_APM_INVALID_SLAVE_PARAMETER (0xC07B0009L)	Invalid slave parameter.
TLR_E_ASI_APM_ACTIVATE_WATCHDOG (0xC07B000AL)	Failed to activate watchdog supervision.
TLR_E_ASI_APM_NOT_ALLOWED_IN_PROTECTED_MODE (0xC07B000BL)	Request is not allowed in protected mode.

Table 158: Status/Error Codes 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

Support

Phone: +49 (0) 6190 9907-99
E-Mail: 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

Support

Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France

Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support

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

India

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

Italy

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

Support

Phone: +39/02 25007068
E-Mail: it.support@hilscher.com

Japan

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

Support

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

Korea

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

Switzerland

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

Support

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

USA

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

Support

Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com